适合开发者 项目经理 CTO&CIO 编程爱好者阅读收藏

Broadview
www.broadview.com.cn

PROGRAMMER 程序员

PROGRAMMER 程序员 2016 精华本

程序员编辑部 编

16大篇章、210篇文章 讲述成功产品背后的 技术、人和事

聚焦技术实践 关注前沿热点 开发者年度必备

中国工信出版集团 电子工业出版社

程序员网址：http://programmer.csdn.net

程序员 2016 精华本

程序员编辑部　编

总策划：孟迎霞

编委会：钱曙光　卢　凯　唐小引　魏　伟　陈秋歌　纪明超
何永灿　仲培艺　郭　芮　付　江　张红月　屠　敏

程序员 2016 精华本

图书在版编目（CIP）数据

程序员 2016 精华本 / 程序员编辑部编. —北京：电子工业出版社，2017.1
ISBN 978-7-121-30747-8

Ⅰ. ①程… Ⅱ. ①程… Ⅲ. ①程序设计－文集 Ⅳ. ①TP311.1-53

中国版本图书馆 CIP 数据核字（2016）第 316811 号

责任编辑：董　英
印　　刷：北京京科印刷有限公司
装　　订：北京京科印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：850×1168　1/16　　印张：40.5　　字数：1620 千字
版　　次：2017 年 1 月第 1 版
印　　次：2017 年 1 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

前　言

助你拓展视野，保持怀疑

1965 年，犹他大学电子工程系迎来了一位计算机课程讲师 Robert S. Barton。每堂课，他都在“摧毁”学生们对计算机的“信念”——他让每位学生了解前人的工作，却不让学生信仰任何一种理论，于是他们不得不学习一切计算机理念，继而又在他的质疑下将刚建立起的信念推倒。学期结束时，不再有学生会执迷于某项技术，因为他们对一切充满怀疑。

怀疑，能让人从新视角看待熟悉的事物，在 Barton 的学生中，不单有 Alan Ashton（WordPerfect 设计者）、Alan Kay（2003 年图灵奖得主）、James Clark（创立了网景和 Silicon Graphics），还有 John Warnock（Adobe 创始人之一）、Edwin Catmull（皮克斯创始人之一）、裴祥风（发明了 Phong 反射模型及 Phong 着色法，并广为 CG 界采用）等人。

或许你正专注于工作中使用的某项技术，某个工具，但请别因依赖而局限自己的视野，《程序员 2016 精华本》为你展示不同行业实践的方方面面，解决相似问题的不同视角，它会帮助你拓展视野，保持怀疑。

不只关注当下，更在乎深刻涵义

Alan Kay 曾说，“计算机”完全是门流行文化，忽视历史，让它成不了一门科学。假如你向一位数学或物理学者打探前辈的故事，他们总能如数家珍，而计算机从业者则不然，大多数人连自己领域中“高斯”和“牛顿”的名字都想不起来。

流行文化只关乎参与和存在感，却不在意过去与未来。然而更深刻的涵义会在只关注当下的文化中消逝，2016 年，这个领域失去了 Peter Naur（2005 年图灵奖得主）、Raymond Tomlinson（被誉为“电子邮件之父”）、Robert Fano（曾与克劳德 • 香农共同开发出香农-范诺编码）、John Ellenby（“Xerox Alto”电脑设计者之一）、Seymour Papert（计算机教育与学习理论先驱）、Marvin Minsky（AI 领域创立者这一）……这些计算机领域缔造者的思考与贡献远非括号中的注释所能概括，然而这些名字中多数甚至从未见诸这一年计算机媒体笔端。

《程序员 2016 精华本》为你记录这一年计算机技术发展，不只关注时下流行技术，也整理行业的历史与思考，不只在今天有意义，也希望对明天的你有价值。

程序员编辑部

目　录

对话大师

2016技术盘点

互联网应用面面观

技术解析开源大数据构造

物联网开发初探

Spark核心技术与实践

走进VR开发世界

人工智能60年，后深度学习时代关键技术进展

容器技术经验谈

云计算与大数据

移动开发新技术探索

基础技术

数据库华山论剑

无人驾驶技术解析

视频直播技术实践

信息无障碍

我们需要一次解决所有问题

——访 wiki 创造者 Ward Cunningham

记者 / 卢鹄翔

在互联网上得到正确答案的最好方式，不是提问，而是写个错答案。

——Cunningham 定律

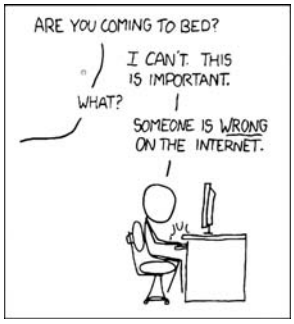

图 1　xkcd 386: 使命召唤

Ward Cunningham 因创造 wiki 而闻名，但在 wiki 诞生后的第二年，有件事就令他忧心忡忡。

从 1995 年至今，C2（c2.com，第一个 wiki）已创建了数万页面，都储存在 Cunningham 自己的服务器。“页面完全归我所有，这种感觉很糟。”wiki 的核心理念本是“任何人都能新添或编辑页面”，但页面却不属于作者，而是由服务器的所有者掌控。Cunningham 痛恨这种“中央控制”——服务器崩溃，数年的积累便可能付诸东流，还意味着贡献者需服从旁人的编辑规则。虽然人们能搭建自己的 wiki 站点，但访问者寥寥，因为大部分读者只汇集在几个知名站点。

Cunningham 将自己的新创造称为 Federated Wiki，在他的设想中，人人都该拥有自己的 wiki，并结成联盟。若想编辑别人的页面，只需点击“Fork”——像 GitHub 那样，将页面复制到自己的 wiki 并开始编辑，而原始页面则留下了你的线索，所有人都能看到，而原始页面拥有者，能决定是否将合并修改——Wikipedia 常迫使编辑者放弃自己的观点，而 Federated Wiki 则能让异见共存，这些观点又相互连接。

搭建自己的 wiki 可能会令许多人望而却步——人们已经习惯将内容发布在 Facebook、Tumblr 这些有人打理的平台，不过 Cunningham 认为“虽然简化是好事，但不该以令用户无助为代价”。他不是空想家，也理解如何简化，自己动手实现了 Smallest Federated Wiki（SFW）——一个用于展示 Federated Wiki 理念的简化版，既可一个命令在本地电脑安装，也可以一键部署于 AWS。他说，其实 Facebook 已经证明，其实每个人都想拥有属于自己的页面和信息流。

Federated Wiki 的另一创新之处是从程序设计中借用了“重构”的概念。与文字“编辑”不同，Cunningham 说，“编辑”意味着作者心中有预想的读者，了解他们的期望，据此做出修改——重构则是对信息大跨度的重新组合，为内容赋予新的意义，现有编辑软件已不能胜任。在 SFW 中，段落能在不同服务器间轻松拖动、释放，重新组合，并通过联盟机制在网络上分享。SFW 也比从前的 wiki 更时尚、易用，例如图片和视频都能拖放置入，通过插件支持 Markdown 语法、MathJax 公式甚至通过 DSL 实现任何你想拥有的功能。

图 2　Smallest Federated Wiki

Cunningham 说：“wiki 原型完成五年后，Wikipedia 诞生了，也许到 2020 年你会看到 Federated Wiki 的硕果。”

当前研究领域

Ward：我将自己工作的领域称为“仪器度量学”。刚工作那会儿，我将计算机系统思维应用在理解电子仪表上。多年后，我以相同的思考方式解读软件系统。作为工程师和科学家，我总为通过“度量”来理解事物而着迷。哪些参数可被度量？哪些易于度量？如何解读，结果又如何改变我们的下一步决定？

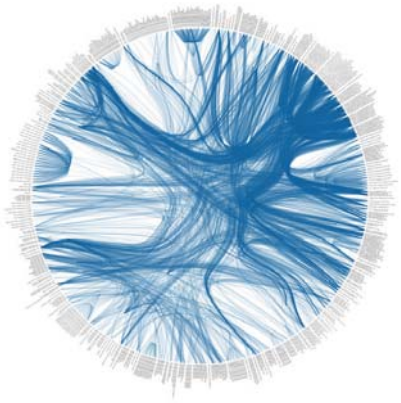

图 3　SFW 站点相互连接（2015 年 12 月）

近来，我参与开发和测试用于度量软件的“仪器”，加入的多个团队都以此为目标。从现实系统中收获的数据，总令我愉悦，我们通过这些数据，测试新“仪器”。通过编写程序创建数据集，我对软件系统，以及用于观察这些系统仪器，理解都更深入。

我专注理解软件系统，特别是用来“理解其他软件系统”的系统。我所供职的公司（New Relic）无论自身，还是其客户规模，都在快速增长。许多工作，都从理解这种快速增长对软件有何种影响开始。

程序生涯与感悟

Ward：我尽力记录自己编写过的每段程序（从 1964 年高中开始），职业生涯轨迹可以划分为以下几个阶段。

驾驭掌握：在学生时代，我编写的大多数程序与“抽象”和表现这些“抽象”有关。直到我将程序展示给朋友，它们才变得完整。

分享知识：我离开研究机构，转而在企业工作。在企业中有诸多束缚，也带来不同期待。不过在束缚之外仍有创造的空间。

位居要职：我持续使用不同语言编程，程序设计从未如此简便，但人们却总能找到办法，让编程变得困难。渐起的声名为我敞开了许多扇大门，而我选择的媒介——编程，却常常令邀请我步入大门的人深感困惑。

我注意到，当人们意识到一处大错，会奋起而战；但倘若是多处小错，却无动于衷——觉得生活似乎本就如此。而这些小错，往往并不能逐个排除，这样做不但无济于事，甚至会令事情更糟。人们对编程的理解也与此相似，小问题盘根错节，没有一个能得到修正。

我动员人们与我为伴，一次解决所有问题。

所以我开始讲一个当一切都已变得更好的故事，故事很长，但我大可一点点讲。朋友们也乐意参与其中，并开始讲述他们自己的故事。故事越来越美好，修复不完美的现状，融入自己的经验。我们从故事中学习，继续讲故事，生活在故事中。终究会有一天，这些故事将成为现实。

我遇到过许多年纪尚不及我一半（Ward 今年 66 岁），却早已觉得编程力不从心的人。我提醒他们，新科技很可能与旧事物别无二致，不过是换了个名字迷惑你。但确实有些新技术，能把老问题处理得更好，并可解决新问题，他们应当加以分辨。

书籍与启发

Ward：对我影响最大的著作，是那些定义了一个空间，并向我们展示作为工程师或者只是普通人，如何在其中创造的书。这些著作“颠覆”，而非仅仅“提升”了我们对于实践的理解。

Dijkstra 的《A Discipline of Programming》（中文版名为《编程的修炼》）向我展示了如何像推导数学证明那样推导程序，由此可以透彻地理解程序，并确定它能正确运行。虽然我也能通过其他机制确保达到相同效果，但这本书讲的是最根本的思维处理过程。

Carver Mead 与 Lynn Conway 合著的《Introduction to VLSI Systems》将计算机设计归至晶体管实现层面，提供了一套易于掌握，又有无限创造可能的法则。虽然受成本制约，某些特定风格的设计眼下并不经济，但其物理本质不变，芯片依旧有效。

George Lakoff 与 Mark Johnson 合著的《Metaphors We Live By》（中文版名为《我们赖以生存的隐喻》），这本书所介绍的认知科学，我发现可以直接应用在理解“对象”上，阅读这本书帮助我更合理地使用它们。

还有一些著作，对我的影响虽不像上面几本那样深远，但在创造 wiki 时，它们讲述的概念一直萦绕在我脑海中。

Leonard Koren 的《Wabi-Sabi for Artists, Designers, Poets & Philosophers》向我展示了无常中的恒久魅力。

Edwin Schlossberg 的《Interactive Excellence》则让我对“社区”与“定义品质”有了清晰认识。

CaffeOnSpark 解决了三大问题

——对话雅虎机器学习平台负责人

记者 / 周建丁

Andy Feng 是 Apache Storm 的 Committer，同时也是雅虎公司负责大数据与机器学习平台的副总裁。他带领雅虎机器学习团队基于开源的 Spark 和 Caffe 开发了深度学习框架 CaffeOnSpark，以支持雅虎的业务团队在 Hadoop 和 Spark 集群上无缝地完成大数据处理、传统机器学习和深度学习任务，并在 CaffeOnSpark 较为成熟之后将其开源（https://github.com/yahoo/CaffeOnSpark）。Andy Feng 接受《程序员》记者专访，从研发初衷、设计思想、技术架构、实现和应用情况等角度对 CaffeOnSpark 进行了解读。

CaffeOnSpark 概况

CaffeOnSpark 研发的背景，是雅虎内部具有大规模支持 YARN 的 Hadoop 和 Spark 集群用于大数据存储和处理，包括特征工程与传统机器学习（如雅虎自己开发的词嵌入、逻辑回归等算法），同时雅虎的很多团队也在使用 Caffe 支持大规模深度学习工作。目前的深度学习框架基本都只专注于深度学习，但深度学习需要大量的数据，所以雅虎希望深度学习框架能够和大数据平台结合在一起，以减少大数据 / 深度学习平台的系统和流程的复杂性，也减少多个集群之间不必要的数据传输带来的性能瓶颈和低效（如图 1）。

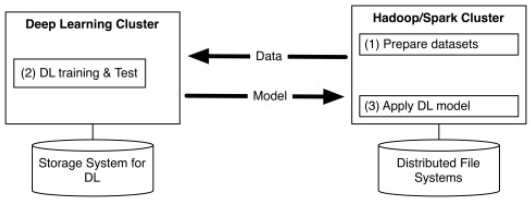

图 1　深度学习集群与大数据集群分离的低效

CaffeOnSpark 就是雅虎的尝试。对雅虎而言，Caffe 与 Spark 的集成，让各种机器学习管道集中在同一个集群中，深度学习训练和测试能被嵌入到 Spark 应用程序，还可以通过 YARN 来优化深度学习资源的调度。

CaffeOnSpark 架构如图 2 所示，Spark on YARN 加载了一些执行器（用户可以指定 Spark 执行器的个数（–num-executors <# of EXECUTORS>），以及为每个执行器分配的 GPU 的个数（–devices <# of GPUs PER EXECUTOR>））（Executor）。每个执行器分配到一个基于 HDFS 的训练数据分区，然后开启

多个基于 Caffe 的训练线程。每个训练线程由一个特定的 GPU 处理。使用反向传播算法处理完一批训练样本后，这些训练线程之间交换模型参数的梯度值，在多台服务器的 GPU 之间以 MPI Allreduce 形式进行交换，支持 TCP/ 以太网或者 RDMA/ Infiniband。相比 Caffe，经过增强的 CaffeOnSpark 可以支持在一台服务器上使用多个 GPU，深度学习模型同步受益于 RDMA。

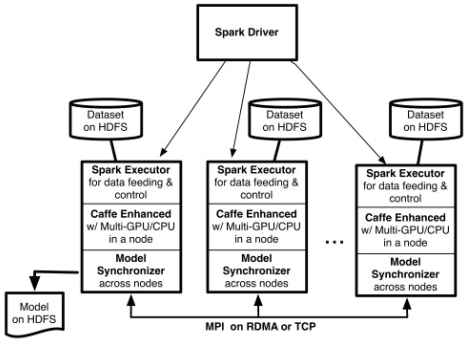

图 2　CaffeOnSpark 系统架构

考虑到大数据深度学习往往需要漫长的训练时间，CaffeOnSpark 还支持定期快照训练状态，以便训练任务在系统出现故障后能够恢复到之前的状态，不必从头开始重新训练。从第一次发布系统架构到宣布开源，时间间隔大约为半年，主要就是为了解决一些企业级的需求。

CaffeOnSpark 解决了三大问题

《程序员》：在众多的深度学习框架中，为什么选择了 Caffe？

Andy Feng：Caffe 是雅虎所使用的主要深度学习平台之一。早在几个季度之前，开发人员就已将 Caffe 部署到产品上（见 Pierre Garrigues 在 RE.WORK 的演讲），最近，我们看到雅虎越来越多的团队使用 Caffe 进行深度学习研究。作为平台组，我们希望公司的其他小组能够更方便地使用 Caffe。

在社区里，Caffe 以图像深度学习方面的高级特性而闻名。但在雅虎，我们也发现很容易将 Caffe 扩展到非图像的应用场景中，如自然语言处理等。

作为一款开源软件，Caffe 拥有活跃的社区。雅虎也积极与伯克利 Caffe 团队和开发者、用户社区合作（包括学术和产业）。

《程序员》：除了贡献到社区的特性，雅虎使用的 Caffe 相对于伯克利版本还有什么重要的不同？

Andy Feng：CaffeOnSpark 是伯克利 Caffe 的分布式版本。我们对 Caffe 核心只做了细微改动，重点主要放在分布式学习上。在 Caffe 的核心层面，我们改进 Caffe 来支持多 GPU、多线程执行，并引入了新的数据层来处理大规模数据。这些核心改进已经加入了伯克利 Caffe 的代码库，整个 Caffe 社区都能因此而受益。

如图 3 所示，CaffeOnSpark 支持在由网络连接的 GPU 和 CPU 服务器集群上进行分布式深度学习（用户可以自行选择向成本还是性能倾斜）。只需稍微调整网络配置，就能在自己的数据集上进行分布式学习。

《程序员》：CaffeOnSpark 希望填补 Spark 在分布式深度学习方面的空白，但 Caffe 设计的初衷是解决图像领域的应用问题，而深度学习的应用不止于图像，能否介绍 CaffeOnSpark 目前具体支持的算法，以及使用场景有哪些？

Andy Feng：正如之前所提到的，Caffe 在深度学习的应用远不止图像。举个例子，Caffe 可以为搜索引擎训练 Page Ranking 模型。在那种场景下，训练数据由用户的搜索会话组成，包括搜索词、网页的 URL 和内容。

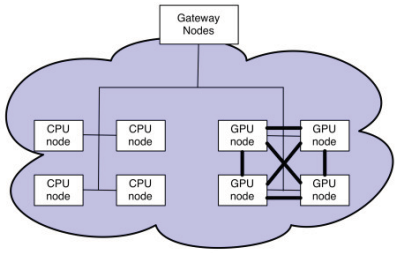

图 3　CaffeOnSpark 支持由网络连接的 GPU 和 CPU 集群

《程序员》：Caffe 在与 Spark 的集成需要解决哪些问题？您的团队克服了哪些主要挑战？

Andy Feng：我们主要做了三个方面的事情：

性能

CaffeOnSpark 是按照专业深度学习集群的性能标准而设计。

典型的 Spark 应用分配多个执行器（Executor）来完成独立的学习任务，并且用 Spark 驱动器（Driver）同步模型更新。二级缓存之间是相互独立的，没有通信。这样的结构需要模型数据在执行器的 GPU 和 CPU 之间来回传输，驱动器成了通信的瓶颈。

为了实现性能标准，CaffeOnSpark 在 Spark 执行器之间使用了 peer-to-peer 的通信模式。最开始我们尝试了开源的 OpenMPI，但 OpenMPI 的应用需要提前选好一些机器构建 MPI 集群，而 CaffeOnSpark 其实并不知道会用到哪些机器，所以我们研发了自己的 MPI。

如果有无限的带宽连接，这些执行器就能够直接读取远端执行器的 GPU 内存。我们的 MPI 实现将计算和通信的消耗分布到各个执行器上，因此消除了潜在的瓶颈。

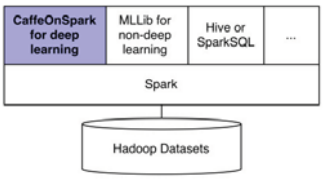

图 4　雅虎希望 CaffeOnSpark 成为 Spark 上的深度学习包

大数据

Caffe 最初的设计只考虑了单台服务器，也就是说输入数据只在本地文件系统上。对于 CaffeOnSpark，我们的目的是处理存储在分布式集群上的大规模数据，并且支持用户使用已经存在的数据集（比如 LMDB 数据集）。CaffeOnSpark 引入了 Data Source 的概念，提供了 LMDB 的植入实现、数据框架、LMDB 和序列文件。CaffeOnSpark 不仅提供了深度学习的高级 API，也提供了非深度学习和普通数据分析 / 处理的接口。详见我们博文的介绍。

编程语言

尽管 Caffe 是用 C++ 实现的，但作为 Spark 的产品，

CaffeOnSpark 也支持 Scala、Python 和 Java 的可编程接口。内存管理对在 JVM 上长期运行的 Caffe 任务是一个挑战，因为 Java 的 GC 机制并没有考虑 JNI 层的内存分配问题。我们在内存管理上做了重大改变，通过一个自定义 JNI 实现。

《程序员》：目前 Spark+MPI 架构相比深度学习专用 HPC 集群的性能差异如何？TCP 方法和 RDMA 方法的差别又是多大？

Andy Feng：我们的设计和 Spark+MPI 实现旨在取得和 HPC 集群相似的性能。这也是我们支持诸如无限带宽连接接口的原因。关于差别，我们还需要做一些扩展性的对照测试。

《程序员》：关于并行训练中常用的参数服务器（Parameter Server），CaffeOnSpark 是如何设计的？

Andy Feng：参数服务器的主要目的是可以建立很大的模型，但目前的深度学习模型还很小，所以参数服务器的用处还不是很大。当然，我们也提供了参数服务器的功能，在需要的时候启用即可。

《程序员》：另一款 Spark 深度学习框架是伯克利 AMPLab 出品的 SparkNet（https://github.com/amplab/SparkNet），此外《程序员》曾刊载过百度团队研发的 Spark on PADDLE（http://geek.csdn.net/news/detail/58867），也是针对 HDFS 数据传输的瓶颈做了优化。您如何看待这些框架的不同？

Andy Feng：CaffeOnSpark 和 SparkNet 都是为了在 Spark 集群上运行基于 Caffe 的深度学习项目。SparkNet 用标准的 Spark 结构，在 Spark 执行器内用 Spark 驱动器和 Caffe 处理器通信。

然而，CaffeOnSpark 在 Caffe 处理器之前使用 peer-to-peer 通信（MPI），避免驱动器潜在的扩展瓶颈。CaffeOnSpark 支持 Caffe 用户已有数据集（LMDB）和配置、增量学习、以及机器学习管道的高级接口。详情参见 https://github.com/yahoo/CaffeOnSpark#why-caffeonspark。

对于 Spark on PADDLE，我简单地浏览了《程序员》的文章，它的框架结构（和 CaffeOnSpark）非常相似。由于 Spark on PADDLE 项目没有公开，不清楚它是怎么和 Spark 生态系统整合的，比如 MLlib 或者 Spark SQL。

《程序员》：能否介绍 CaffeOnSpark 社区的规划，对开发者有什么要求？

Andy Feng：我们正在组织一个活跃的 CaffeOnSpark 社区。我们欢迎社区成员尝试使用 CaffeOnSpark、提出问题、查找缺陷以及贡献解决方案。我们会采用 GitHub 审阅流程来确保所有代码都是高质量的。

《程序员》能否介绍 CaffeOnSpark 开源之后的应用反馈？接下来它的功能改进和问题解决重点包括哪些？

Andy Feng：目前 CaffeOnSpark 支持同步分布式学习，我们计划在不久的将来支持异步分布式学习。我们的 MPI 也能通过改进来处理系统崩溃问题。

如我们预计，CaffeOnSpark 开源之后，每天的访问量都在增长，包括一些国内的用户。很多人为他们在使用过程中遇到的一些问题向我们寻求帮助（估计用于图像类的应用比较多）。

例如，CaffeOnSpark 发布的 API 主要是 Scala API，一些用户提出了 Java API 的需求。这个请求是合理的，因为 Spark 支持 Scala、Python 和 Java 三种语言的 API。我们会根据需求的强烈程度规划改进，但预计 Python API 很快就会出来。另外一个是 UI 方面的问题，我们后续也会逐步改进。

同时，用户之间也在交流他们使用的经验。因为雅虎能够自己测试的实际环境比较有限，针对一些问题我们也只能给出建议，用户相互交流会找到一些方案，让 CaffeOnSpark 的发展更好。

《程序员》：您对 CaffeOnSpark 在 Spark 社区的发展有什么样的期待？

Andy Feng：我们希望 CaffeOnSpark 促使 Spark 社区致力于深度学习领域。在雅虎，我们已经看到深度学习在大数据集上取得的显著成果，也期待着听到社区成员传来更多的好消息。

务实至上：“PHP 之父”Rasmus Lerdorf 访谈录

记者 / 卢鹄翔，张新慧

“PHP 之父”Rasmus Lerdorf 性格直接坦荡，措辞简练精辟，字里行间透着一股“务实至上”的精神气。在参加“PHP 全球开发者大会”前夕，这位“实干家”接受了《程序员》采访，分享了自己的编程感悟。

工作重心

我参与的项目总与用户直接相关。我曾多年担任雅虎工程师，负责连接数亿终端用户的基础设施，这些基础设置仍在服役。而如今，我在 Etsy 也是负责连接百万用户的后端基础设施。技术其实只是解决问题的工具，是抽象的锤头、锯子，并没什么了不起，而真正振奋人心的是用技术提升了百万人的生活品质。

“问题”为指引

比起“计算机科学家”，“工程师”的称谓更得我心。论这两者的区别，我认为后者更专注于解决眼前的问题。之所以开发 PHP，并非因为我喜爱编程或语言设计，反倒是因为不喜欢。1993 年的那套编程把式，让我没法轻松迅捷地解决 Web 问题。于我而言，只有当遇到困难，才会翻翻书，查查资料来充实自己，技能不是为了提高而提高，我的每一个决定都是以解决问题为中心的。

经验和教训

我犯了很多错，有些事后才意识到；但也取得过好于预期的成果。最重要的经验是：解决 Web 问题的确应该从一开始就专注相关的整个生态系统。20 年来，针对 Web 问题的解决方案层出不穷，而质量却参差不齐，没几个能构建起完整的生态圈，并为普通人所用。

PHP 7 何处费思量？

开发 PHP 7 那最后 10% 最费时，也最无趣。不过强大的新

功能和性能突飞猛进，还是让积极心态占了上风，也激励了整个团队，帮我们很快熬了过去。不过，每次更新都有做不完的测试、解决不完的平台问题，调查不完的诡异边缘情况，看不完的漏洞报告，没个尽头。

吸纳新人

我知道前不久 Emacs 的版本控制系统从 BZR 换成了 Git，不过对于吸纳新贡献者，我觉得它们其实平分秋色。Git 近来更受欢迎，方便蜻蜓点水式的添砖加瓦。不过对于长期的忠实贡献者，版本控制系统无关紧要。良好的文档和方便新贡献的流程才重要呢。

假如能重新设计 PHP

假如时光能倒流，肯定有我希望能改进的地方，比如区分 Keyword 大小写。刚开始 PHP 不过是种 HTML 模板语言。九十年代初，人们争论 HTML 标签是该大写、小写还是大小写混合。我不想争来争去的，就把 PHP 的模板标签做成不区分大小写的，这个做法至今还在沿用。

JavaScript 在吞噬其他语言吗？

PHP 和 JavaScript 的演进几乎同步。我与 Brendan Eich（JavaScript 设计者）是同一时期开始的，他的重心显而易见是客户端，而我则是服务器。如果你写客户端应用，除了 JavaScript，别无选择——浏览器支持哪种语言就得用哪种；但如果重心是服务器就很不一样了。

写客户端代码者众，所以会 JavaScript 的人多，而如今它在服务器这厢也开花结果了。但 JavaScript 跟 PHP 一样只是解决方案之一，而非唯一，这样挺好。就像我刚说的，语言只是解决问题的工具，不是受人膜拜的宗教。如果眼前有问题，而你更倾向于 JavaScript，那么它就是最佳选择。

编程原则

只要有效、安全、够快，就发布，然后解决下个问题。三者缺一不可，否则就要回头检查代码，好抓紧时间解决下一个问题。

未来展望

关于编程语言，我还真没想过这些工具未来会经历什么，我更关心的是它们能否解决当下的问题。拿 Etsy 举例，作为手工工艺品网站，它能在富有的买家和穷苦艺术家之间牵线搭桥，让他们摆脱贫穷的窘境吗？我们的基础设施能帮助其他公司去应对同等重大的问题吗？我们的解决方案是否强大到一转眼客户的问题就去无踪了呢？这些才是我关心的。

过去，我不喜欢编程，现在还是不咋喜欢。我只喜欢以解决问题为中心，这点永远不变。

科研的秘诀

——对话微软研究院负责人 Peter Lee

记者 / 卢鹄翔

四十年前，位于加州 Palo Alto 的 Xerox PARC 发明了个人电脑和 Bit-Map 显示器、GUI、WYSIWYG 和桌面出版、面向对象编程、激光打印机、以太网、Peer-Peer 和 Client-Server 网络，以及“半个”互联网。而这些至今仍影响我们生活的发明，仅由 25 位研究员，在 5 年中完成，每年的项目经费，仅约合当下的 1 千万美元。

创造为何在那个时代涌现？如今计算机科学研究高效运行的关键在何处？未来是否能带给我们更多期许？ Peter Lee 作为微软全球资深副总裁，执掌着当今最多产的计算机科研究机构之一——微软研究院（Microsoft Research，简称 MSR）中的体验与新技术部门（New Experiences and Technologies，简称 MSR NExT）。带着这些疑问，我们试着在与他的对话中，寻找解开答案的线索。

《程序员》：你所负责的 MSR NExT，是个怎样的部门，与我们通常了解的计算机科研有哪些不同？

Peter Lee：“研究”可以围绕两种方式组织：一种是基础领域，例如机器学习、计算机安全、网络等，这是种重要且有力的研究方式。而在 MSRNExT，我们选择了另一种——研究并非围绕领域展开，而更聚焦于特定目标和问题。例如我们能消弭网络间用户的语言障碍吗？社交媒体如何保障隐私？是否能开发出新的处理器技术延续摩尔定律？一组研究员围绕这些具体的目标展开研究。

为确保能真的解决关键问题，需要新的组织方式。为此，我们投入巨大的精力，厘清研究目标，确定会对世界和微软都能产生巨大影响的目标，然后聚焦于此。这与人们通常说的“进一步理解计算机网络或者机器学习”完全不同——虽然这些研究仍在微软研究院中进行。

MSR NExT 成立于 Satya Nadella 就任 CEO 之后，甫一年余，Satya 希望 MSR 成为微软创新的中坚力量，赋予我们更多权限，同时又寄以厚望，MSR 在微软的整体战略中从未如此重要。而今我们同在飞驰的过山车上，既兴奋，也深知责任重大。

《程序员》：你选择项目还是研究员？

Peter Lee：项目创意从研究员中来，因而我们非常看重研究员。我们要做的是将这些创意分享，让模糊的想法更具体更合理，以及确定从何处开始。这个过程有点儿像风投，挑选投资哪个初创企业。

《程序员》：你与许多杰出的研究者朝夕相处，他们身上有哪些共同点，与一般人不同？

Peter Lee：与众不同的之处有两点，第一，他们甘愿长时间全身心投入一项研究，他们能耐心专注一个问题许多年，在所有杰出的研究者身上，我都看到过这一点。

我想起好友 Eric Horvitz，他是位世界级的 AI 研究者，其研

究领域为 Bayesian Graphical Models，在他眼里，整个世界都能用这种概念表示。有次，我们在一家精致的饭店进晚餐，犹豫开胃菜该点什么时 Eric 说，何不用 Bayesian Graphical Model 算算，或许能找出最可口的那种。结果我们点了菜单上列出的每一种开胃菜，摆满了整桌子，现场开始计算——你能看出来他对科学有多痴迷。

第二点，杰出的研究者对与人合作能敞开胸怀，他们愿把自己的成果倾囊分享，而不是有所保留。我常见到奇妙的研究，始于拥有完全不同背景的研究者合作。

《程序员》：去年我曾请教 Ivan Sutherland 如何做出好研究，他说有三个要素：首先要有个好问题，其次需要资金支持，最后，也是最难寻觅的，是需要一位睿智的管理者。你曾是研究员，而如今领导着拥有多位图灵奖得主在内的研究机构，如何才能让这所研究机构高效运转？

Peter Lee：我刚到 DARPA 担任研究经理时，也曾恳请 Ivan Sutherland 指教。他坦诚无私，专程坐飞机赶来华盛顿，与我共度了两天时光。他也给我三个建议，与给你的完全相同。不过他额外告诉我一条规则——信赖人，而非提案。假如有一位屡获成功的研究者，寻求经费和支持，就该为他提供——哪怕你并不相信提案本身。这点很重要，在我的职业生涯中，屡获见证。管理科学研究是项不同寻常的工作，当研究员需要支持时，我需要恪守准则，不以提案是不是个好点子来判定。我通常会不断提问，挑战研究员和他们的提案。

《程序员》：通过哪些问题？

Peter Lee：例如为什么采用此方案，而非其他？又或者为什么我该相信，这项研究能令世界变得更好？还有些是更具体的技术问题。提问的意义在于迫使对方思考，重新检验他们的假设。在这个过程中，方案往往会优化，有时还能发现更有潜力的方向。不过设想你是位研究员，兴高采烈地找我聊聊自己的新想法，却不断收到质疑。好多人的第一反应都是自己被冒犯了。你可能会想，这个愚蠢的家伙怎么那么多问题？不过随着时间推移，便会逐渐理解我提问的初衷。

而一旦确认立项，我就得闪开了，不再以任何方式影响课题发展。通常我们会提前敲定一个时间点，在项目推进几个月甚至几年之后才碰下头，不过在那之前，我绝不干涉。聚集一批杰出的研究者，为他们提供资助，然后闪到一旁去——这是 Sutherland 教给我的。

例如在 MSR，Catapult（将 FPGA 用于数据中心，以提升效率的同时降低能耗）是项耗资巨大的研究，负责人 Doug Burger 需要争取用于测试和部署 Catapult 芯片的资金。我的第一个问题是：过去二十年，许多人曾提出使用 FPGA 的想法，但结果均不尽如人意，为什么这次我该信你？

“为什么你的想法现在很重要？”是我常提出的第一个问题。研究员往往就此陷入冥思苦想——为何是现在？之所以难回答，因为这需要预测未来——或许时机尚早，研究不能产生广泛的影响，又或者有赖于其他技术成熟。

第二个问题是：假如研发出这种 FPGA，并在数据中心投入使用，它能解决哪些棘手的麻烦？这是个关于“应用场景”的问题。Doug 思索了一年多，终于找出了具体场景——对 Bing 的搜索 Web 页面做排序。在许多人看来，场景似乎不难寻觅，但真要找到恰当的，绝非一蹴而就——就像你明知屋里有蚊子，但等开灯去找，它们却都藏起来了。

接下来的问题是“最简可行产品”（MVP，Minimum Viable Product）该怎么做，来证明你的观点？见微知著，研究不可能总是从宏伟的设计开始。为了回答这些问题，Doug 殚精竭虑，但也正因为如此，Catapult 项目才得以成功，其成果如今已在我们的数据中心广泛部署。

《程序员》：有种观点认为，昔日的研究机构如 Xerox PARC、Bell Labs，创造胜于今日——即便是在有限的资源条件下，一如 Alan Kay 在“The Future Doesn't Have to Be Incremental”中提到的。你觉得是这样吗？

Peter Lee：我很喜欢这个问题。MSR 颇有 XeroxPARC、Bell Labs 基因，因为众多研究员曾在那里工作。

而作为 MSR 管理者，我也渴望和梦想自己所在的科研机构能做出改变世界的创造，就像 XeroxPARC 创造了个人计算机，Bell Labs 创造了晶体管那样。不过你也知道，晶体管发明 7 年后，世人才逐渐了解这项创造的价值，并在更久之后获得诺贝尔奖；Xerox PARC 也耗费多年，才让世人认识个人计算机，而它自己却失去了将其推向大众的机会。我有时在想，会不会我们已经做出了这样的创造，只是尚不自知，说不定十年之后会有分晓。

Alan Kay 的观点使人深思。现如今，科技公司对科研及研究员的重视程度前所未有，微软、Google、Amazon、Facebook、百度都在吸纳越来越多博士生，因为企业明白，这些博士研究生能创造出巨大的商业价值。然而仔细思量却会发现，这些创造是渐进的（Incremental）——令 Bing 的搜索结果更准确，或者让 Windows 系统更上一层楼，诚然可嘉——能为企业带来可观的商业收益。但在 MSR，我也会提醒研究员，帮助公司创造利润固有其价值，但还应时刻铭记，我们真正的使命和内心渴求，是通过自己的创造改变世界。但意识到这一点却绝非易事，因为当你的提案被快速实现，并应用到数以亿计用户使用的产品中时，似乎能带来更直接的满足感。

这是长久萦绕在我脑海中的问题，我总是鼓励别人当树雄心壮志，希望我们足够勤奋也足够幸运，有朝一日做出这样的成就。

无人机的背后

宾夕法尼亚大学工程学院院长 Vijay Kumar 专访

记者／徐威龙

提到无人机，有人也许会想到巡航在海上的大型的军用武器，而近年来，随着消费级无人机的技术革新和价格优势，这类无人机开始被大众所熟悉。无论是摄影还是送货，只要一提起无人机，人们马上会联想到大疆（DJI）、Parrot（一家总部

位于法国巴黎的无人机及无线产品制造商）、3D Robotics（由前《连线》杂志主编 Chris Anderson 和时年 20 岁的 Jordi Munoz 创建，主打北美市场）、AscTec（德国 Ascending 公司旗下的无人机品牌）这些生产商。

然而，在美国宾夕法尼亚大学有这样一间实验室，早在 2006 年就开始了无人机相关技术的探索，至今他们的无人机算法和技术在农业、救灾、建筑、娱乐等方面有着广泛的尝试。相对于常见的摄影或物流类的无人机，他们所研究的无人机应用的场景更多样、处理的问题更复杂、涉及的技术领域和更为宽泛。可以说，他们的每一个新进展，都引领着整个无人机领域的技术变革，同时也是推动无人机和人工智能领域发展的杠杆。

近日，《程序员》杂志在 CCF－GAIR 全球人工智能和机器人峰会上专访了美国宾夕法尼亚大学工程学院院长，IEEE、ASME Fellow，美国国家工程院院士，Vijay Kumar。他就自己实验室最新的研究成果与我们进行了交谈。

除了摄影，无人机还能干什么？

Vijay Kumar 是位印度裔机器人科学家，1983 年毕业于印度理工学院坎普尔分校（IIT），后进修于美国俄亥俄州立大学，1987 年获得机械工程博士学位。Vijay 是世界上第一批研究无人机的科学家之一，早在 2006 年，Vijay 和他的团队便第一次成功开发出了四轴飞行器原型，并积极推广无人机在多个实用场景下的应用。十年后的今天，无人机产业早已变得炙手可热，市场规模实现了百亿美元的增长，有人估计，到 2020 年，无人机市场将达到 250 亿美元。尽管市场越来越热，但学术界和产业界之间，依然存在一定的距离。这也让 Vijay 坦言，他并不关心无人机市场，眼下自己的重点，依然放在无人机的四个核心研发问题上：体型、安全、智能、速度。虽然这四点看似简单，但要是一一展开，无疑会让我们看到一幅关于无人机的更大的画卷。Vijay 的最新研究成果，就是对这四点最好的诠释。

这些成果包括以下几类。

■ 自主小型飞行器（Autonomous Micro UAVs），这种飞行器可以利用 GPS 在复杂的三维环境中巡航，可以用于搜寻工作和精细农作（Precision Farming）。这种无人机的重量只有 20 克，通过传感器来规划路线和监控当前环境，传感器包括 IMU（Inertial Measurement Unit）、摄像头、激光测距仪、测高仪等。

■ 飞行器矩阵（SWARMS，Scalable sWarms of Autonomous Robots and Mobile Sensors），在矩阵中，每台飞行器都是以自主（Autonomic）的状态完成任务，在没有中央控制系统的情况下，仅根据傍边飞行器的反应而作出调整，从而达到协作的目的。

■ 微型生物机器人（Micro Bio Robots），这些微型机器人只有 10 ~ 100 微米，由生物传感器和马达制动。他们可以被用于药物探索、微创治疗以及微型装配。同时，他们还可以被应用在生物传感器和生物电路的合成，帮助行成“传感——驱动——通信”的闭环。

■ 飞天手机（Flying Smart Phones），智能手机是当下最常见的高性能、低成本处理器和传感器集成，用无人机搭载智能手机，相当于拥有了一个给力的 CPU/GPU、两个摄像头、一块电池、GPS、Wi-Fi、蓝牙模块、IMU、电信数据接收器，以及相应的内存。这无疑给飞行器赋予了更好的性能。

通过上述研究，无人机可以被应用在很多场景，例如可以放进大楼作为报警器，一旦有入侵者，马上报警；在建筑内寻找生化／煤气泄漏，或者把飞行器送进倒塌的楼房或核反应堆，来探测放射强度；室内建模，在陌生的空间内构建实时地图，甚至连一些室内细节都可以反映出来；搬运重物；搭建建筑龙骨（目前，根据 Vijay 一位学生开发的算法，飞行器矩阵可以自己建造矩形的龙骨），飞行器根据对周围飞行器的探测，可以自行选择用什么材料、用在哪里、什么时候用。

除此之外，Vijay 研究的飞行器还有很多实际用途，例如精细农作，通过热感摄像机和普通摄像头捕捉的数据，分析每株作物的特定需求，并且将结果反馈给农户，比如作物 A 需要浇水、作物 B 需要施肥、作物 C 需要除虫等。通过这一方式，无人机还可以计算每株作物的果实数量，从而帮助农户对收成进行预估，以及找出染上疾病的作物。

仿生学在无人机领域的应用

对仿生学的应用是 Vijay 研究中最重要的思想之一，也是很多创新技术诞生的基石。在上面提到的无人机四个要素中，体型大小会对其他因素造成不同程度的影响。通过对蜜蜂的观察，Vijay 发现，这种小型飞行生物可以很好地跟环境互动，即使相撞也不会受伤。同时小体型意味着惯性更小，速度更快，受到撞击时也可以很快恢复。但小体型的代价却是搭载的传感器更少，单个飞行器能做的事情也有限。

为了弥补这一缺憾，Vijay 和他的团队发明了飞行器矩阵，即通过一组飞行器来配合完成某项任务，这也是 Vijay 团队无人机研究中最重要的概念。为了让一组飞行器更有机地配合，他们再次借鉴了仿生学规则。通过对切叶蚁亚科的盘腹蚁属（Aphaenogaster）进行观察（如图 1 所示），Vijay 总结了三个重要的自然生物协同工作的原则。

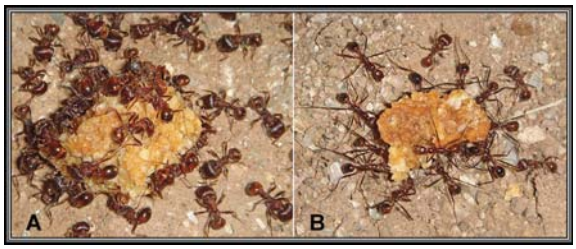

图 1　切叶蚁亚科的盘腹蚁属通过协作搬运重物

■ 独立自主（Decentralized），蚁群不受任何“中央命令”的控制，每只蚂蚁都是按照自己的本能做事；

■ 临近感知（Neighbor Sensing），单只蚂蚁的行为根据自己“邻居”的行为作出调整；

■ 匿名（Anonymity），单只蚂蚁对自己的“邻居”是谁并不清楚，因此无论“邻居”换成谁，都不会对自己的行为造成影响。

受到上述自然规则的启发，Vijay 设计了原理类似的飞行器矩阵，这种矩阵被称为 Swarm（如图 2 所示）。在 Swarm 中，每台飞行器都装有本地传感器，并且拥有本地通信和计算能力。两台相邻的飞行器通过传感器控制彼此的距离，传感器以 100 次 /s 的频率将数据发送到处理器，并由处理器将解决方案传送给电机。在图 2 的示例中，Swarm 利用邻近感知原则，可以在障碍物前转换阵型（Formation），由于匿名原则，在一个 Swarm 中，

即使抽掉（或增加）了几台飞行器，阵型依然可以保持原样。

图 2　Swarm 无人机矩阵

而由于每台飞行器都是独立运算行动的，在另一个例子中，飞行器通过自身对环境的探测，完成了一系列高难度的动作，比如在狭窄的通道前转体，钻过通道后迅速恢复原样；以及钻过向上抛出的铁圈等（如图 3 所示）。这两个例子的原理在于，人们先为飞行器预先编入飞行轨迹，但飞行器需要在预设的轨迹上加一个动作，这需要它们先积聚动能再改变方向，再回到预设的轨迹上来。它们知道怎样完成每段独立的轨迹，然后把每一小段的轨迹组合起来。

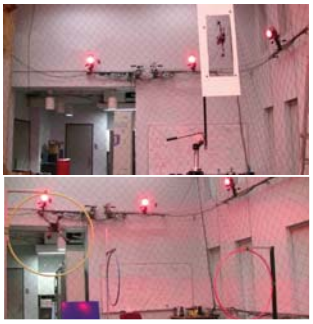

图 3　飞行器通过自身对环境的监测，完成一系列高难度动作

为了更好地理解 Swarm，Vijay 通过一个理论来进一步说明，他称这个理论为 Consensus，Consensus 解决的是在 Swarm 中到底该怎样形成阵型（Formation），哪台飞行器该飞到哪，矩阵如何保持一致性的问题。Vijay 举了一个例子。假设你身处一个 5 个人的房间，每个人脑子里都想一个数字（事先不能交流），这时 5 个人都想到同一个数字的概率小到可以忽略不计，而这时你把你想到的数字告诉“邻居”，“邻居”再把自己听到的数字相加再除以人数。比如 A 说 1、B 说 4、C 说 10，当你问 C 的数字是几的时候，C 会说 5（1+4+10 = 15，3/15）。如果你们重复这一步骤，那么几轮之后，你们就会得到相同的数字了。这就是为什么每台无人机都有自己的飞行路线，但最终却能协同起来的原因。

尽管在充分训练的情况下，Swarm 已经拥有良好的表现，但 Vijay 也坦言，随着飞行器的数量增多，Swarm 的组织也越难，原因有二。其一是随着飞行器数量的增加，单个飞行器对“邻居”动态的感知更难，需要掌握的数据也更多；其二在于，当不同特征（Traits）的飞行器同时加入编队后，对阵型的优化影响很大。

Vijay 团队曾做过一项实验。实验中，他们为不同的飞行器赋予不同的特征（Traits），使具备相同特征的飞行器成为一个“种群”（Species），让不同“种群”的飞行器组成一个阵列，去完成某项任务。结果发现，种群的基数越大阵型就越难被优化。

无人机，走出实验室要面临更多的挑战

今天，无人机在现实世界中的真实应用案例还相当有限，尽管 2011 年，Vijay 的无人机曾被用来日本地震灾后的探测工作（如图 4 所示），但要想让无人机在更多的任务中发挥作用，依然面临着不小的技术挑战，外部世界的复杂性远远超过实验室的模拟，除了上文提到的 Swarm 之外，无人机在室外甚至是灾难现场工作，还面临着风速、无 GPS 定位、悬停电量消耗、陡坡降落等难题。由于 Vijay 研究的无人机是“独立”且完全“自动”的（Autonomic），即使要完成从 A 点到 B 点这样简单的动作，都要穿过一个 12 维的空间。

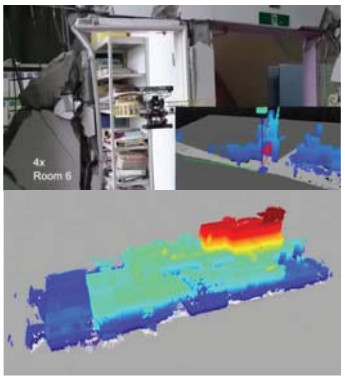

图 4　2011 年在日本一所被地震震塌的房子内构建三维地图，为了节省“体力”无人机由一台地面机器人载入房间，之后再进行飞行探测

Vijay 的方法是把这个 12 维空间翘曲成一个 4 维空间（如图 5 所示），这个 4 维空间由横轴、纵轴、竖轴、旋转轴组成。飞行器在这个 4 维空间中以 10 ~ 100 次 / s 的频率规划“最小化加速轨道”（Minimum Snap Trajectory），同时通过对位置向量、导数、速度和加速度的推导，创造一个优雅的飞行曲线，并且绕开障碍物。

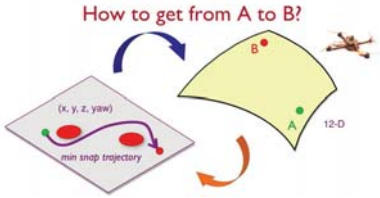

图 5　无人机的飞行算法

然而当飞行速度达到一定的量级之后，上面的方法就不利于避障了。由于无人机上的传感器无法探测 20 米外的范围，这时如果无人机以 20m/s 的速度运行，一秒钟后就会飞到一个完全陌生的环境或撞上障碍物。这时就需要建立一个二维地图，在二维地图中找到安全轨迹作为矢量集合，飞行器可以在这个矢量集合中去调整“最小化加速度轨道”。

下面我们来看看 Vijay 团队在实际应用方面的一些关键技术研究。

地图构建

地图的构建分为两种，一种是在未知环境，且传感器数量和信号都不好（尤其是无法使用 GPS）的情况下创建拓扑地图；另一种是构建 SLAM（Simultaneous Localization And Mapping）。关于第一种，Vijay 和他的团队开发了一套创建拓扑地图的算法工具，通过 ROS 呈现（机器人操作系统）。这一算法需要先对自由空间进行覆盖，然后建立起该覆盖环境的基本维诺图（Voronoi Graph，如图 6 所示）。

而 SLAM 的构建则要复杂一点，在讲解技术原理之前，我

们不妨简单地看一看这个近几年在机器人、VR/AR 领域经常用到的技术。简单来说，SLAM 是指当某种设备（如机器人、VR 设备等）来到一个完全陌生的环境时，它需要精准地建立时间和空间的对应关系，并能完美地回答以下一系列问题：我刚才在哪里，现在在哪里？我看到了什么？现在看到的和之前看到的有哪些异同？我过去的行走轨迹是什么？我现在看到的环境是什么样子，和过去相比有怎样的变化？我的轨迹抖吗？我的位置飘吗？我还能跟踪到自己的轨迹吗？如果我丢了应该怎么办？我过去建立的对环境的认识还有用吗？我能在已有环境的抽象里快速对现在的位置进行定位吗？

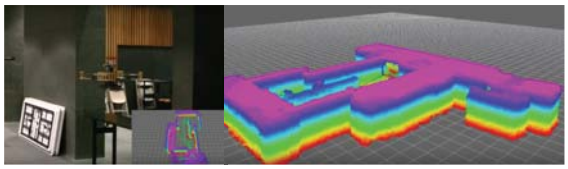

图 6　陌生环境下室内拓扑地图的构建

在 Vijay 的实践中，他们采用了一种叫做“直接视觉 SLAM”（Direct Visual SLAM）的方法。与传统的用来构建“稀疏重建”（Sparse Reconstructions）的功能性算法相比，直接视觉 SLAM 更适合建立“半密集重建”（Semi-Dense Reconstruction）和“全密集重建”（Fully-Dense Reconstruction），如图 7 所示。Vijay 告诉我们，他们目标是在一个标准 CPU 环境下，将视觉数据和惯导数据连接起来。整个过程被拆分为三个线程，第一个线程利用视觉数据和惯导数据做非线性优化，对摄像头的移动做出判断，这一过程以帧速率运行，然后构建出半密集地图（Semi-Dense Map）。第二线程则需要构建出一个拥有高梯度区域（High-Gradient Area）的半密集地图，目的依然是对摄像头位置进行追踪。第三线程则以较低的帧速率构建出该场景的全密集重建。

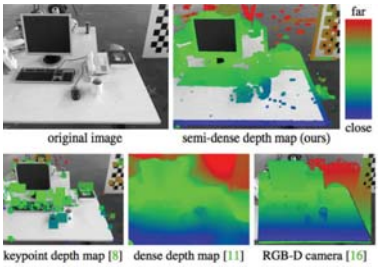

图 7　半密集重建与全密集重建图

无 GPS 环境下，微型无人机矩阵部署系统

只装有少量传感器的微型无人机矩阵，是否可以在恶劣的环境下工作（比如无法使用 GPS 等）？为了解答这一问题，Vijay 团队在矿井和大风的户外进行了试验，他们开发了一套叫作“基于视觉的稳定系统”（Visual-Based Stabilization），这套系统被部署到了多个应用模式中，包括：主从无人机矩阵（Leader Follower）、Swarm 矩阵在监控环境下的部署与稳定、合作无人感知测量等。实验中，这些飞机的控制、运动规划、稳定性和轨道规划都有不俗的表现。

高速无人机在陡坡的降落

救援型无人机需要在灾难现场长时间实时传输信息或构建地图，然而续航时间短仍然是个重要的短板。为了解决这一问题，Vijay 和他的团队，希望通过让无人机随时降落来“保存体力”，包括降落在各种质地和各种角度的平面。Vijay 为微型无人机设计了“抓器”，它可以牢牢“抓住”不同的平面，还可以抓取重物。下图展示了高速飞行的无人机在陡坡降落的情况，目前 Vijay 的无人机可以在 90° 的陡坡实现高速降落（如图 8 所示）。

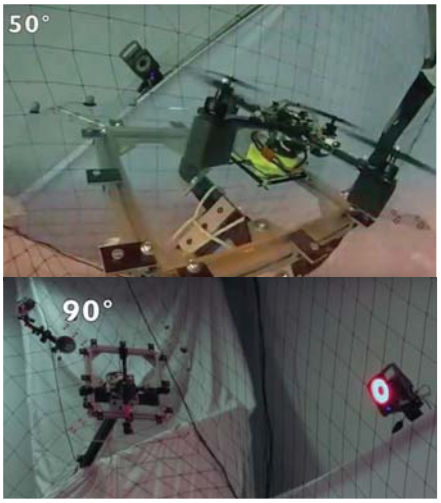

图 8　高速飞行的无人机在 50° 和 90° 的平面降落

这个“抓器”不是一般的“爪子”，而是一个真空吸盘，通过微型真空泵来驱动。他们对吸盘的设计和它与放气、激活力度、最大抓举力之间的关系做了很多次测试，最终这一吸盘做到了既可以抓取各种物体，又可以在陡坡上降落。

为了节省“体力”，Vijay 和他的团队还做过其他尝试。比如 2011 年他们让一台机器人背着无人机进入一栋坍塌的大楼，进到大楼呢，无人机才开始工作，用了两个半小时绘制出了大楼内部的地图。

近景遥感技术在农业上的应用

通过近景遥感（Close-Range Remote Sensing）技术对柑橘属果树进行检测，并对患青果病的果树进行识别。在佛罗里达，青果病是威胁果树健康的主要因素。在同一景深下，通过测量黄色偏振光的反射比，可以找出患病的果树。此外，他们还制定了一套标准用来测量患病果树树叶中的淀粉沉积。在获得这些信息之后，他们再加上一个景深维度，和机器学习算法，识别的准确率达到 93%。不仅能识别患病的果树，连青果病的传染和早期受到感染的果树也能分辨。

与 Vijay Kumar 对话实录

程序员：你最近的研究并不太依靠外部设施，飞行器是完全“自动”（Autonomic）的，而用到的技术却谈不上创新，你怎么看？

Vijay：目前我们所用到的技术，比如动作捕捉等确实称不上新技术，这类技术我们早在 2008 年就有所尝试。但技术背后的算法却是新的，理论也是新的，比如上面提到的陡坡降落等。新的理论让“旧”的技术发挥新的作用。

程序员：你用手机作为传感器集成，比起载有等量传感器

的无人机，具体可以减重多少？

Vijay：一般一个载有等量传感器的飞行器大约重 3 千克，悬停的时候会更重，电缆、传感器、外壁的重量占 45% ~ 50%，要 240 瓦才能带动。而一台 iPhone6 只有 129 克，只相当于原来十分之一的重量。

程序员：既然无人机对重量的要求如此苛刻，那么执行救灾任务时很难载人离开？

Vijay：的确。悬停是最大的问题，悬停时每千克消耗 200 瓦电力，要是载一个 50 公斤的人，每次悬停就要消耗一万瓦以上的电力，这不是个小数字。但也不是没有解决办法，我们可以看看客机，无人机也可以像客机那样用飞机燃料、汽油或柴油当作燃料。

程序员：你认为无人机要想真正普及，技术是最大的门槛吗？

Vijay：我不这么认为。技术创新固然重要，但很多需求我们已经知道该如何实现了。但商业创新现在依然处于摸索的阶段。无人机要想实现普及，需要由商业创新来推动，这是我不擅长的。你看 Uber 和滴滴出行这两家公司，他们的崛起并非因为他们在技术上有很大突破，而是因为商业创新。

Alan Kay 和他的浪漫愿景

文 / 杨硕

在纽约时代广场的威斯汀酒店餐厅里，我见到了约定已久的 Alan Kay，他等待时正在看书。Alan Kay 因为在面向对象编程方面的贡献获得了 2003 年图灵奖，他从犹他大学毕业后在斯坦福大学人工智能实验室工作，之后在施乐公司的 PARC 实验室工作了十年，这段时间，他发展了计算机图形界面（GUI）并发明了面向对象编程。之后 Alan Kay 在苹果和迪士尼做研究，有人称他是个人电脑之父，或者计算机图形界面之父。

如果你从事计算机领域研究，尤其是交互领域，就需要知道这个领域的历史和愿景，而 Alan Kay 的愿景是最清晰最浪漫的，他能想象未来应该是什么样，并且努力让其实现。

他所研究的领域远超计算机，总能从更高层次看问题，他的想法又总是很宏大，也许你一开始并不理解他在说什么，但回头看总会有新收获，他的每篇文章和每个演讲都令人深受启发。

他讲话风趣且深刻，似乎随口说的一句话就能放进名言库里，你肯定听过这句话：“预测未来的最好方法就是创造未来——The best way to predict future is to invent it”。

早期生活 ——艺术和科学的融合

Alan Kay 出生于 1940 年，父亲是位设计义肢的生理科学家（Physiologist），母亲是位音乐家。他从小成长在一个科学和艺术共存的环境，他说自己至今仍不区分“科学”和“艺术”。他三岁就能流畅阅读，他说这“既是种幸运，也是种不幸，等上学已读完了 150 本书，因此我知道老师总是欺骗我。”

他本科在科罗拉多大学（University of Colorado at Boulder）学习生物，还没毕业就加入了美国空军，在那里接触了计算机，并通过资质测试成为一名 IBM 1401 程序员。从空军退役后，他回到科罗拉多大学获得了数学和生物学位，那段时间他也是职业爵士乐吉他手。之后 Alan Kay 在犹他大学（University of Utah）获得了电子工程硕士和计算机科学博士学位。而当时犹他大学计算机专业的研究领域主要由 J.C.R. Licklider 所管理的 ARPA 资助。

研究生阶段，Alan Kay 接触了 Ivan Sutherland 的 Sketchpad，深受其图形界面和交互方式影响，Sketchpad 是一个开创性的图形界面编程工具，可以用触控笔在显示器上画图来制作计算机模型。Sketchpad 完全改变了 Alan Kay 对编程和计算机的看法。1968 年，他见证了 Doug Engelbart 著名的“演示之母”（Mother of All Demo），Alan Kay 如此描述这段经历：“就像是看着摩西打开了红海，给我们展示了极有潜力的新大陆，并指引了到达那里所需要跨过的河流和海洋”。深受 Engelbart 影响，Alan Kay 记住了图形界面、超链接，以及鼠标等想法。同样对他有深刻影响的还有 Seymour Papert 和 Marshall McLuhan，Seymour Papert 让小孩子学习电脑的项目启发了 Alan Kay 为儿童设计的想法，而 Marsall McLuhan 的《理解媒介》（Understanding Media）让他意识到计算机是一个比古登堡印刷术还要重要的新媒介。

在这些影响下，Alan Kay 提出了一个在当时（1968 年）极为超前的概念，一个带有触控笔的平板电脑，称其为 Dynabook。由于当时技术不可能实现，他用现在设计师们常用的方法——纸板做了一个模型。虽然技术在彼时尚未成熟，但 Alan Kay 知道摩尔定律，他相信解决技术困难只是时间问题。

从犹他大学毕业之后，Alan Kay 来到斯坦福人工智能实验室（Stanford Artificial Intelligence Lab），而他对人工智能并不感兴趣，在那里他仍然研究如何为孩子做计算机。1971 年，Alan Kay 加入了施乐 PARC 实验室（Xerox PARC），不久成为了那里的核心成员。

Xerox PARC——活在未来

Xerox PARC 是由施乐公司在 1970 年成立的一个远离公司总部的研究所，目的是为了研究“未来的办公室（The office of the future）”Alan Kay 负责其中的 Learning Research Group。

Xerox PARC 的宗旨是找到最聪敏的人并且让他们做自己想做的事。但这不单单是雇佣一些聪明人那么简单，还要营造相应的社区、文化和支持创新的环境。PARC 当时的管理者 Bob Taylor 有一个理念就是“活在未来（Living in the Future）”，根据摩尔定律，电脑成本会每个 18 个月下降一半。所以他决定不考虑成本，花钱购买性能最强的设备。他希望能够活在未来去创造未来。施乐公司能为这些新想法买单，而摩尔定律会解决成本问题——Alan 认为至少该活在未来 5 年，如果可以，应该活在未来 10 年，去创造那时的计算机以及网络，而且不论建造什么，一定要亲自使用。

Alan Kay 是 PARC 愿景的来源，他在那里继续研发 Dynabook，很多想法都集成在了 Xerox PARC 研发的 Alto 计算机中。而如今的笔记本、平板电脑都能在 Dynabook 设计中里找到线索。

有段时间，施乐公司高层总是想让 PARC 成员预测公司未

来的“走势”。在其中一个令人恼火的会议上，Alan Kay 生气地回答道 “预测未来的最好方法就是去创造未来”，而这句话后来成为 PARC 的信条，并被乔布斯多次引用。

为了帮助人和计算机交互，Alan Kay 和他的团队创造了 Smalltalk 编程语言。Smalltalk 最初的设计是一个图形化编程语言，但很快变成完整的编程环境，包括 Debugger、Object-oriented 虚拟内存、编辑器、屏幕管理和图形界面。其中有我们现在熟悉的图标、弹出菜单、下拉菜单、滚动条和层叠窗口等，Smalltalk 是 Dynabook 最终界面的原型。Alan Kay 希望让他的程序成达到这样的目标 “简单的事情应该简单，复杂的事情应该成为可能”（Simple things should be simple, complex things should be possible）。

Alan Kay 的界面设计哲学基于心理学家 Jerome Bruner 的学习理论，而 Bruner 又基于 Jean Piaget 的认知理论。因为 Alan Kay 是为儿童设计，所以他在做界面设计时将这些学习理论作为重要准则。在这一点上，他和 Doug Engelbart 的想法不同，Engelbart 是在为成人尤其是知识工作者而设计。而 Alan Kay 选择跳过一代人，直接为下一代设计，因为他希望下一代能更好地使用计算机这个“媒介”，而不需要强迫改变孩子们的思维方式，他说“小孩子本身就出生在一个新的范式中”，因为对小孩子来说一切都是新的。

1983 年 Alan Kay 加入苹果继续从事研究，1997 年他的团队挪到了迪士尼的 Imagineering 部门。五年后，他成立了 Viewpoints Research Institute，一个致力于为提高“powerful ideas eductation”的研究机构。而如今，他在 Y Combinator 的赞助下发起了研究项目 YC Research（https://ycr.org）。项目里有一个叫 HARC 的小组——Human Advancement Research Community（人类进步研究社区），汇集了包括 Bret Victor 及 Vi Hart 等世界一流研究人员。

Alan Kay 的观点

和 Alan 聊天的过程中，他提醒我，不要让计算机替我们思考。我们需要想的问题是，如果每个人都有一个仆人，他们会做什么？我们还能存活下去吗？罗马人到最后无法生存，是因为他们让希腊人替自己思考。工具是把双刃剑，一方面，它们在某一方面给你优势，让你不需要做琐碎的事；另一方面，工具却减弱了你的知觉。美国作家梭罗曾有句话“我们会成为我们工具的工具（We are becoming the tools of our tools）”——当第一个跨大西洋电缆接好时，别人问梭罗怎么想，他说自己担心收到的第一条消息会是欧洲的公主买了顶新帽子。

计算机和古登堡印刷术

什么是媒介？解释媒介的最好例子就是古登堡印刷术。文艺复兴时代的欧洲，印刷机到来之后，人类第一次可以大规模互相交流，这永久地改变了社会结构。各种信息和革命性想法能比以前更自由地超越国界传播。这个革新获取了大量的民众并且威胁了政治和宗教的权利——文化的急速普及打破了少数人（神职人员）对教育和学习的垄断，培养了逐渐兴起的中产阶级。

印刷机的在文化上产生重要影响的原因并不是其技术（墨水和金属类型），而是印刷作为一个媒介在特定的方式上增强了人类思考能力。印刷术是文明和受教育社会出现的直接原因。它使得社会的得以自我管理。伽利略和牛顿的科学著作只有在有印刷术的社会中才能出现，而美国宪法，也只有在有印刷术的社会上才能让联邦党人文集在报纸上公开讨论最终形成宪法。

McLuhan 关于媒介的重要观点是“一个新媒介的最初内容永远是一个旧媒介（the initial content of any new medium is always an old media）”——例如书写的内容是讲话，印刷物的内容是书写，而电报的内容是印刷物，早期广播的内容是报纸新闻，最初电视的内容是广播，最初电影的内容是舞台剧。

受 McLuhan 影响，Alan Kay 认为“印刷机”的发明是人类进入新文明的重要转折点。它不仅是台机器，还使得知识传递的成本极大降低，让知识从少数僧侣传递到普通人手中，从而引发了科学革命。

而印刷术古已有之，它的发明可类比于真空管或晶体管，而印刷机的发明则是集成电路。这个类比可以帮助我们理解计算机的意义。与印刷机一样，计算机也是媒介的转折点——它可以将动态信息、知识、科学模型更快更广地传递到普通人手中，它的影响将远大于印刷媒介。

理解 Alan Kay 观点，最好的办法要属 Bret Victor 在他“2013 年阅读链接”中的解释了（http://worrydream.com/Links2013/）。当 Alan Kay 在讲计算机时，是把计算机当成媒介（Medium），就像印刷机一样，而不是把计算机当成技术（technology）。

当你想要理解 Alan Kay 的话语，试着不要去想计算机技术，而是去构想一个不同的社会。在这个社会，人们可以在计算机媒介提供的新维度下进行自如思考和辩论。不要去想“写代码（coding）”，那些是墨水和金属类型的问题，已经过去了；也不要去想“软件开发者（software developers）”，这与中世纪抄写员类似，只有在非文化社会中才合理。而是该去思考，模拟现象，模拟情景，获取对于非线性过程的直觉。在这样的社会中，每个受教育的人都可以做这些事。正如我们今天在书写的媒介下阅读或书写复杂的逻辑论证一样简单和自然。

“阅读”曾经只是那些少数神职人员（牧师、僧侣）的特权，他们负责给大众传达不容置疑的神圣真理。而今天，阅读成为了每个人都在做的事。想象一个世界里，科学不再是少数人的特权，向大众传达不容置疑的真理，而是每个人做的事情——这就是 Alan Kay 想要创造的世界。

在 Alan Kay 的演讲中，我最推荐 2009 年他在我的母校 UIUC 演讲，题目是《Normal consider harmful》，在这个演讲中，他提醒我们如何逃出正常平面（Normal Plane），眼界的力量，以及扩展视野的重要性。

“媒介即信息”

我向 Alan Kay 问了一个困惑已久的问题，McLuhan 在《理解媒介》里说的“媒介即信息”是什么意思？维基百科上的解释是，人们理解一个讯息时会受到其传播方式的影响。我的理解为媒介本身的发明改变了人类感知世界和交流的方式。媒介的发明比媒介所传播的内容更有影响力。McLuhan 也由此提出了“媒介所传递的信息本身是另外一种媒介” ——比如，电报的内容是印刷，印刷的内容是写作，而写作的内容是说话。

Alan Kay 解释道，理解一个媒介里的信息，最重要的是我们需要成为什么样的人。任何想要接收植入在媒介中信息的人

必须先要内化媒介（internalized the medium），让媒介成为自我意识的一部分，这样才能从中抽取出信息。当他说“Medium is the message”时，他的意思是，你必须要首先要成为媒介才能吸收其中的 Message。这对于工程师来说很好理解，收音机要接受一个无线信号，首先要做的就是模拟发布者才能调解信号。

类似地，书中的信息并非印在纸上的文字，而是在读者阅读时，他们所发生的变化。在某种信仰之下，人的信条和其自我无法区别。因为他们从来没有学会如何思考——必须要把自我分离出来才能思考。否则，任何不同的观点对人来说都是一种威胁而不是一件可讨论的事。

观点远胜 IQ

Alan Kay 提到一个非常重要的观点“观点抵得上 80 点 IQ”（A point of view worth more than 80 IQ points）。他举了三个例子，达芬奇有很高的 IQ，在他的时代构想了很多发明，而亨利福特出生在合适的年代，他拥有足够知识，得以制造出汽车改变了人们的交通方式，而达芬奇却不能实现自己的发明。所以“知识（Knowledge）”远胜 IQ。而“知识”的问题是，我们大部分人知道很多知识，但知识并不总是有益的。而有的知识在某些时候是对的，另一些时候又是错的。比“知识”更强大的，是能够改变我们思想和观念格局的“观点”，比如牛顿的微积分为我们提供了一个强大工具，好像多了一个大脑。另如今普通高中生，也可以完成牛顿时代之前最聪明的人都不可想象之事。因此观点 > 知识 >IQ。

新闻与新（News vs. New）

另一个观点是“News vs. New”，也就是“新闻与新”的对比。每当个新想法出现时，你会遇到两类东西，一类是“新闻（News）”——对我们已知事物的一种递增。你可以在 5 分钟内弄明白一个新闻，我们生活中看到的大部分信息都是此类，比如某某电影上映了，或者某某开始选举等。

而还有一类是“新（New）”，从定义上看，它不是我们任何已知的东西，而是那些你前所未知的，或者是要完全改变之前观念之物。印刷术出现时，只为神职人员所用——印圣经，这算是 News。而当有人用印刷术制作成书，传播不同的思想时，才算是 New。从 News 到 New 的转变是漫长的，因为大多数人不接受新事物，大部分人都生活在特定的范式（Paradigm）中，对他们来说，正常（Normal）胜过一切新奇怪想法。所以学习一个新的想法，几乎需要拥有创造那个新想法一样的想象力。

你必须意识到，“正常（Normal）”并不是真实，而仅仅是一种构想，一种社会普遍认同东西。观点的强大之处在于它能帮助我们跳过正常的平面，从不同的方式看世界。那么如何提高观点呢？一个办法就是去许多不同国家，那样你就会发现，诸多你认为理所当然的事，在另一个国家却完全不同，甚至是疯狂的。你能看到上千种看待世界的不同方法，而所有这些都是人们脑海里的虚构故事而已。还有一个办法是“思考”，提醒自己“正常不是真实”。他在演讲的最后引用 Susan Sontag 的一句话：“所有的理解来自于不接受这个世界表面的样子”（All understanding begins with our not accepting the world as it appears）。

尽管 Alan Kay 创造的个人电脑、面向对象编程、视窗系统已经得到广泛使用，但他觉得自己远未成功，他仍在不停歇地描绘一个更好的世界。他的图灵奖获奖演讲标题是《尚未发生的计算机革命》（The Computer Revolution Hasn’t Happened Yet）。他想让每个人都可以利用计算机媒介构建模型，模拟场景研究科学，他想引导人类进入新的文明层次。

2012 年，有人问 Alan Kay，哪个产品最接近他当初设想的 Dynabook。他说到，大部分人用计算机并不做任何有用之事，几乎所有人使用电脑都是为了方便地处理过去的媒体（比如读新闻、写邮件、看视频等）。使用电脑的几十亿人中几乎没有人通过在计算机做模拟（Simulation）去学习新东西，所以还没有任何类似于计算机文明的例子出现。现在的文明状态仍然是相当于用印刷机生产圣经，而且还在模拟僧侣们的手写字体的样式（似乎这是对我们设计领域现状很好的比喻）。

他认为如今 iPad 提供的服务糟透了，因为它违反了个人电脑的第一原则，也就是创造（creation）和消费（consumption）对称。苹果找到了合理的价格，生产了人们愿意为之付费的产品，但并没有让计算机成为更高效的生产工具——有违计算机先驱们对个人电脑的浪漫愿景。

每当你创造一个工具，既做了一个增强器又做了一个假肢。汽车在一定程度上增强了我们，同时在另一方面弱化了我们的身体——有了汽车以后我们必须主动运动才能保持健康。文字书写是一个革命，而苏格拉底却埋怨书写让人健忘，因为能把东西写下来，人们就不愿用脑子记忆。那如今我们使用的科技产品呢？或许需要每个人重新思考。

Alan Kay 谈 OO 和 FP

译 / 王江平

理解“对象”的历程

关于“对象”（object）的理解，我经历了几个不同阶段。

第一阶段是 50 年前，在 ARPA 研究生院的开始几周，我的几种专业背景，数学、分子生物学、系统和程序设计等，与 Sketchpad、Simula 和 ARPAnet 这些东西产生了碰撞。这使我观察到，既然一台计算机可以分解成多台虚拟计算机，相互间持续通信，于是你便可以：

- 完全保留表达式的威力；
- 随时为任何可建模的东西建模；
- 无限地伸缩，而不限于已有的分解计算机的方式。

我喜欢这些。分时运行的“进程”已是这种虚拟机的体现，但因为开销太大还缺少实际的通用性（那就寻找方法消除这些开销……）。

尽管可以为任何事物（包括数据结构）建模，对我来说这还远远不够。真正了不起的是为极端的可伸缩性需求提供松耦合的封装和消息机制（以一种类似生物和生态系统的方式）。

第二个阶段包括在“Lisp 世界”中掺入 Lisp 本身，

McCarthy 关于机器人和时态逻辑的思想，在 ARPA（尤其是在 MIT）进行的 AI 工作，以及 Carl Hewitt 的 PLANNER 语言。有一种思想：对象可以像服务器一样，且可以是面向目标的，使用 PLANNER 类型的目标作为接口语言。

第三阶段是 Parc 的一系列 Smalltalks，试图在 Parc 的 Alto 系统（128K 内存，一半用于显示设备）所能实现的和未来必然需要的功能之间寻找一种实用的平衡。这项工作是与 Dan Ingalls 和组里其他一些天才的同事合作完成的。理想主义的小宇宙一直让我不爽，但从实用角度结果不错。

第四阶段（也是在 Parc）是重新深入探讨时态逻辑和“世界线”（world-line）思想（后面细说）。

第五阶段是再次严肃地思考可伸缩性并重新审视“协作语言”（比如 Gelernter 的 Linda），将它们看作以通用的发布和描述方式进行描述匹配，从而实现松耦合的一种方法。我仍然喜欢这种思想，并希望看它发展到对象可以真正“协商意义”的程度。

McCarthy 的时态逻辑：“时间中的真正函数”

我对这一切的思考方式大部分都可追溯到上世纪 50 年代的 John McCarthy。John 是一位出色的数学家和逻辑学家。他希望自己能做严密一致的逻辑推理——同时希望他的程序和机器人也能做到。机器人是个关键：因为他想让机器人有时在费城，有时在纽约。按常规逻辑这会有问题，然而 John 针对“事实”成立时能表现“时帧”（time frame）的所有事实额外添加了一个参数，从而修正了这一问题。这就创建了一种简单的时态逻辑，将“事实集合”显现为世界线的层层堆栈。

这很容易泛化为“变量”、“数据”、“对象”……的世界线。从个体角度来看，值的“历史”替换了“值”，从系统角度来看，整个系统被表示为每当系统处于两次计算之间时它所处的稳定状态。Simula 后来采用了这一思想的一个弱化但却实用的版本。

应当提一下 Christopher Strachey（编者注：1916 年— 1965 年，生于英国英格兰伦敦汉普斯敦，计算机科学家。他是指称语义最早的提出者之一，也是编程语言设计的先驱，发展了编程语言 CPL）——Lisp 和 McCarthy 的伟大粉丝，他认识到通过始终使用（来自前一时帧的）旧值来产生新值并安装在新的时帧中，很多种编程模型都可以统一起来且更加安全。认识到这一点是因为他首先观察到 Lisp 中“尾递归”是多么干净利落，然后又看到这样的尾递归写成某种循环的形式更易理解：循环中包含类似赋值的语句，其中右边从时间 t 中取值，被赋值的变量则存在于时间 t+1 中（且这样的赋值只允许一次）。这就统一了函数式编程和同时模拟时间和状态的“类命令式”编程。

也要提一下 Ashcroft 和 Wadge 设计的 Lucid 语言（编者注：一种数据流语言，用于非冯模型编程），该语言扩展了 Strachey 的许多思想。

另外，数据库中的“原子事务”也值得一看，思想很类似，只是粒度更粗——从来没有破坏什么，也不用竞争条件，新的版本以一种非破坏性的方式创建出来。其中有了版本的历史。

“时间是个好主意”，这是关键的一点——我们想要它，想用安全、合理的方式处理它——而这些方式中绝大部分（如果不是全部的话）都可以是处于稳定世界线状态序列之间的纯函数式事务。

“刚刚计算来的稳定状态”非常有用。它再也不会改变——因此它代表了系统模拟中的一个“版本”——且可以在产生下一个稳定状态的函数式转换中充当源数值。它还可以充当数据源，为那一刻的世界创建视觉效果。历史信息则可用于调试、撤消、回滚等需要。

在这种模型中，“稳定状态之间时间并不存在”：“时钟”只在新状态完成时走表。就程序而言，CPU 本身并不充当时钟。对于具有内在、干净时间模型的确定性关系，这种思想带来一种非常简易的处理方式。

出于很多理由（但没有一个好的）这种维持安全的方法在上世纪 60 年代输给了在命令式编程中使用竞争条件，然后再通过可怕的、可能导致死锁的信号量来保护它们的做法。

某些时候，任何了解 Lisp 且对对象间的消息传递感兴趣的人都必然“运用”并注意到一种对象（一个 lambda 的“东西”，可以是一个闭包）可以捆绑到一组参数上（看起来有点像消息）。如果一个人知道 Lisp 1.5 是如何基于新式的延迟绑定参数求值来实现的，理解还会更深入。这里指的是 FEXPR，而不是 EXPR——未求值的表达式可作为参数传递，之后再求值。这使得不太优雅的“特殊形式”四处充斥，它们本来可以写成漂亮的惰式函数。

使用前述的时态建模，可以松掉“求值—应用”的耦合，通过安全的消息传递来获得时间层之间的函数式关系。于是，由于我一直喜欢从系统模拟的视角看待计算，我便把“对象”和“函数”看作两种互补的思想，它们并不冲突。

术语一旦成为宗教或更加严格的选择与风格，便会失去它们的意义。这里说的自然是“面向对象”和“函数式编程”这两个术语。我不会将“函数式编程”的一般概念跟任何特定语言等同起来。我坚持认为“函数思想”是一种可靠的映射。类似的，我也不会把“面向对象”的一般概念同任何特定语言等同起来。如今这两个术语被“殖民”了，意思也变了。

还有一个大问题是“对象”和“抽象数据类型”的混淆，以及对“数据”和“赋值语句”的固执。如果真正强化了封装，对象就可以根据需要处理设计参数（包括保存历史信息）。

分布式环境中的对象

有朋友问我现在如何理解分布式环境中的对象和系统模拟的，是否依然认为有可能构造一种系统，它既有互联网那样可以有机增长的规模，又有类似于虚拟时间的良好的、可预测的语义？

好问题。这里面有好几组问题及其权衡关系。二十世纪 70 年代 Dave Reed（编者注：美国计算机科学家，UDP 协议设计者）最初思考的是面向整个互联网的操作系统应具有什么样子。在他提出的许多有趣想法中，其中之一是通过由虚拟时间（pseudo-time）组织起来的分布式克隆计算来处理长延时和海量潜在用户的问题，然后慢速的互联网仅用于输入和偏差同步。这就是我们本世纪 00 年代早期在 Croquet 中实现的东西，那时互联网上一次典型的、还算不错的 ping 来回大约 80 到 100 毫秒。这已经好到可以无需任何服务器而创建《魔兽世界》这样的大规模并行游戏（甚至飞行仿真游戏）了，只要有玩家正在用着的那些机器就够了（分布式）。后来的版本做得更多、更全面。

插一句，去掉实时图形和交互的 Croquet 便自动提供了一种分布式的面向对象数据库。虚拟时间是背后的大理念，保护着数据库中的分布式原子事务。

而且多年来人们一直在讨论。它能否实现？需要在哪些领域做多少工作？等等。

我们在 Parc 做过的与对象有关的工作就是 ARPA/Parc 社区中发生的网络思维的一部分，它最终产生了思考更高层网络实体的倾向。“网络层实体”的一个相当不错的、过渡性的实现是 Gerry Popek 在 Parc 花了一年时间思考“网终系统应有的样子”之后，和他在 UCLA 的团队一起完成的 LOCUS 系统。那是建立在异构机器类型之上的一种迁移式的负载平衡思想，与虚拟时间概念形成高度的互补。

我希望看到富有才华的团队把第二个问题再过一遍。对可伸缩规模的适应性很难事先预测，最好实际去实现它。对于虚拟时间，如果你了解它却不使用它，任何情况下我都觉得有点疯狂，而如果不了解它，那就有点业余了。在这两者之间，则是以各种“不二法门”为特征的宗教（而这跟任何科学都背道而驰）。

Alan Kay 谈读书

译／郭韵

1940 年，Alan 出生在美国马萨诸塞州，1 岁时随父亲举家迁往澳大利亚。他 3 岁就可以阅读，童年在一间存有 6000 本书和大量绘画作品的大房间度过。二战爆发后，Alan 全家又迁回美国。一直以来，Alan 都对自己的阅读量和独立观点感到骄傲。这种眼界和思想使他反感学校的传统教育，从而陷入了一段与老师和教育体制的“长期斗争”之中。他一度认为学校只能存在一种观点——那就是老师的观点或者课本的观点。5 岁时他已经开始用“稚嫩的声音维护自己的观点”，这使得他产生了对年轻一代教育的责任心，这种责任心伴随着 Alan 的一生，以至于他后来许多大师级创意的最初理念都是出于对儿童的关爱。

Alan Kay 曾公布了一份书单（http://www.squeakland.org/resources/books/readingList.jsp），这份书单早在 90 年代初就已经成形，本篇文章是他就这份书单和自己的阅读心路与读者所进行的深度交流。

书单是你为学生而列的吗？

Alan：这张阅读清单实际上并非为我的学生而作，它从 90 年代初期就已经慢慢成形。当时 Anderson Consulting（现为埃森哲咨询公司）的人向我讨教应该阅读的“十大”书籍。对于想扩大阅读范围的人来说，只拣选十本几乎是不可能的事。所以我将首先挑选的十本书籍仅仅作为介绍的开始。而阅读所谓的“坏书”其实也是增加知识的一部分，它们能更好地帮助我们加深对“好书”的理解与领悟。

你阅读的时候会“全部吃透”还是“有的放矢”？有哪些帮助定位好书的渠道可以推荐？

Alan：从某种机缘巧合的意义上来说，我开始上学前就养成了阅读的习惯，并且一发不可收拾。现在数不清我阅读了多少本书，可能不少于两万本（据我了解，有些朋友的阅读量比我还大）。阅读的流畅度的确是很重要的因素，因为真正值得吃透的书并不多（正如一句话所言，“大多数想法要么平庸，要么更糟糕”，这很大程度上是因为绝妙的想法难得一见，而且它们还要在我们身边这个充斥了各种恶俗的所谓常识的环境里孕育生长）。读三年级的时候，我开始努力尝试将所阅读的内容铭记于心（如果读一遍没记住，我只好重读）。因为阅读高效等原因，我几乎读完了所有的书籍。通常我同时阅读几本书。现在我不会像更年轻的时候一样整天整天地看书了。阅读会上瘾。我也读书评，其中值得一提的是 1970 年代出版的《全球概览》（The Whole Earth Catalog），里面包括了成千上万的阅读“工具”，例如几百本带有精简评论的书籍。这些书籍大多数都不容错过（《全球概览》里的书籍均为帕罗奥托研究中心图书馆里收录的第一批书籍）。有些朋友希望我能给每本推荐的书都附上扼要的简介（加了注释的阅读书目其实非常有帮助），其实通常我会依赖直觉（并使用其他各种微妙的感觉）来判断（编者注：读者可以通过 2013 年出版的《Cool Tools》了解当年《The Whole Earth Catalog》的风格）。

说起《全球概览》，据说这本书的发起人 Stewart Brand 是在一次嗑药之后想到的主意。作为 60 年代“迷幻革命”（Psychedelic Revolution）的亲历者，一位技术发展的积极推动者，你是否认为从那个年代起，迷幻药对当今的技术甚至社会的发展都有着比较大的影响？

Alan：“60 年代”的意义远大于一个“时代背景”，在加州尤其如此——当时各种思潮相竞迸发：对种族平等的虎丘、越战，以及各种不同的思想。帕罗奥托与湾区的其他地带充斥着所谓的“反正统文化”，也为多元化的创作背景奠定了基础。

我认为这为当时正在发生的事物带来了直接影响——某种鲜明的熏陶，如技术的改变与计算机。当时那里大多数领军人物，尤其是帕罗奥托研究中心的大多数研究员都对毒品直言不讳。不过帕罗奥托研究中心的传统氛围更偏向喝酒。不过，他们也尝试了很多毒品以外的“醒脑”方式，如冥想、自我催眠、水面浮动隔离舱等。

小举动往往有大影响，这原因就多了。

你提到至今大约读过 2 万本书，这个数字是怎么得来的？

Alan：我并没有估算的好办法——当时我也懒得边读边数（阅读不是比赛）。过去很多年，我都保持了每周阅读十本书的习惯，这是当时图书馆每周外借的上限。这些书用不了多久就读完了。现在我 76 岁，过去 73 年我都在勤勉地阅读。我的家庭图书馆大概有 13000 本书，里面还没包括我小时候阅读的书籍，现在我大概保持每周四本的阅读习惯。

所以，迄今我的阅读量大概是两万册。

但人的阅读速度不一样，80 岁跟 30 岁的阅读速度也有差别，你提到的那些一生能读完 23,000 本书的人是不是都有某种“速读”能力？

Alan：是的，的确有这个可能性。这在某种程度上而言，

就像培养出一种看到乐谱就能弹唱的能力一样。这个能力的确不错，因为要心领神会才能做到，就像在 Prima Vista 乐团（将电影音乐会纳入表演节目的四重奏弦乐团，也是罕见的为默片谱曲配乐的组合之一）的演奏中那种得心应手的感觉。

要想做这一点，工多艺熟是关键！在训练过程中，将不同的短期和长期记忆融会贯通，是另外一个关键。Betrand Russell（译者注：英国哲学家、罗辑学家、历史学家、数学家、作家）在他的一生中，就读完了 23,000 本。

书该有系统地读还是随机去读？

Alan：在国会图书馆有超过 2300 万本图书，优秀的阅读者可以在一生中读完 23,000 本已经不错了（我知道阅读数目超过这一数字的人并不多）。因此，我们可以思考在一生的阅读计划中读完 23,000 本的十分之一。当然我们会很希望错过的书籍都用处不大，也不值细读。我觉得很难一言概括，你可以花很多时间去研习如 Linux 这样的庞大系统，而无需先掌握许多电脑方面的知识。因此，读书时要抱着“学海无涯，生命有限”的态度。

阅读的一个大前提就是选择“何时何地阅读哪本书籍”。还有就是挤出时间做更多阅读，涉猎要广。

你怎么记住书里的内容？

Alan：之前很多人问我是如何记住所有书里的内容，其实我很难总结，因为无论过去还是现在，记住书里的内容牵涉到很多因素，这些因素要共同发挥作用，但另一方面，这些因素之间的联系又不是很紧密。我之前提到过一个方法，只记重要的内容。

可以想象一下，如果你偶然在电视看到了二十多年前看过的一部电影会有什么感觉。你花多久时间才能反应过来自己看过这部电影？你是否能很自然地想起下一个桥段是什么？然后再想想自己初次看这部电影的时候，压根儿没想到会在二十年后，别人用电影的几个画面来问你还记得多少。

过去多年有大量研究探讨视觉回放的完整度与细节。最好的回想莫过于通过一帧画面或其他感官触觉（气味是个催化剂）而突发“倏然想起”的感觉。

你听说过马库斯·图留斯·西塞罗（译者注：Cicero，古罗马著名政治家、演说家、雄辩家、法学家和哲学家）在参议院演讲时提到的“思维宫殿”吧，他会在脑海中徜徉于自己的别墅，而他提到其中一些物件时，又仿佛身临其境，描述得栩栩如生。这套理论同样适用于思维的产生。大脑中处理普通思维（Kahneman 的“系统 2”）（编者注：相关概念参见《Thinking, Fast and Slow》）的部分得到释放，让其中产生的想法能够“自由配置”（如不同图像与声音，均可同时存在）。由此产生的许多联想都是各种比喻和类比。

几乎在所有情形下，我们都倾向特别记得住某些事情，要记得其他事情就难多了（尤其是那些当时印象不深，又难想起的事）。但似乎感官记忆中的大部分都更倾向于记得“混合的事情”，而不是更少见的另类之物。

三年级时，我约莫发现了其实“读比想更快”，阅读后要花更多时间才能想通。回顾过去，我想这就是我们在尝试解决问题时所做的背景思考——阅读的道理也一样，这也是音乐中视唱或视谱的原理。

同类道理还有很多，这里讲最后一点。虽然我很小就接触了音乐，但较晚才学习传统键盘（管风琴），这让我有机会让自己手脚并用训练看谱演奏。其实，有好几年对我而言都是痛苦的记忆，尤其是在我那个年纪。其实，看谱演奏的道理跟阅读和记住文本非常相似（除了多了一些需要通过习得才能形成肌肉记忆）。

在我的阅读过程中，最重要的转变就是学会从“字面含义”之外，看到更多的东西，能将刚刚看到的意思运用出来，同时探索里面的延伸含义，并在几秒后也同样运用这些新含义。基本上这就像一个逐步拓展缓缓向前的过程，谁都可以学会，但学起来就不是那么容易（至少对我而言挺难的）。

如果还能将中间的缓冲发展过程与更久远的记忆联系起来，你就更能记住它了，这段过程就像记忆中的丝丝线索。大多数音乐家都有某种双重记忆（他们记忆音乐要比记忆肌肉运动更容易）。这一道理在文本阅读与文本记忆中也同样适用。最简单的方法就是“放松”。

你读书时有做笔记和写阅读总结的习惯吗？

Alan：阅读时我不做笔记，也不做回顾总结。但我会用笔记本记下自己在解决问题时或刚刚出现在我脑海中的想法。这并非是受所谓想法的局限，只是暂时把它们搁置在次要位置，让更多想法得以酝酿。我偶尔会阅读这些笔记，但一般不看。记录这些想法就像是把它们放在胃里存储，稍后再细细消化。

家长一看见自己的孩子读“没用的书”就很反感，而这些“没用的书”却恰恰就是一些人从小读的。小孩子读侦探小说、爱情小说没什么不妥，只要能帮助你去探索生活之外的那个更大的世界。

Alan：很同意。我不认为“垃圾小说”就真的那么“垃圾”。工多艺熟，阅读大量有趣的垃圾小说也未尝不可（我就试过）。对我而言，真正的“垃圾”是那种文不对题，挂羊头卖狗肉的书。垃圾小说中也有很多佳作。

在论文《A Personal Computer for Children of All Ages》中，你介绍了 Dynabook，还说人们可以把这篇论文当做科幻小说来看。你从科幻小说中汲取过什么灵感？

Alan：20 世纪 40、50 年代，我热衷于科幻小说——基本“有书必读”，不过这一热情在 60 年代后慢慢消减了。部分是由于自然科学渐渐取代科幻小说，成为了我阅读的主线，另一原因是在许多领域，科学和技术超越了科幻，也由于我最爱的作者后来的作品数量也渐渐变少了，内容也不再精采。

有哪些“编程之外的编程书”可推荐？

Alan：有许多。例如《分子细胞生物学》（Molecular Biology of the Cell）、《形态合成笔记》（Notes on a Synthesis of Form）等。还有之前书单中其他著作。

你对 Gregory Bateson 在人类学、精神病学、控制论等领域的交叉研究是否有了解？你是否读过他的《Steps to an Ecology of Mind》？

Alan：这是一本我本应放到清单上的书。我对 Gregory（编者注：英国人类学家、语言学家、符号学家。1940 年代他帮助将系统论 / 控制论扩散至社会 / 行为科学领域，晚年致力于发展一种认识论的“元科学”，统一系统论的各种早期形式，《Steps to an Ecology of Mind》为他的观点合集）的了解不多，对他的女儿 Mary Catherine 了解也较少。在这一领域，其他人也同样值得

关注，例如 Gordon Pask、Heinz Von Forester 等。

从更广泛的意义上，“有趣的人不会因为你赞同或否定都会照样有趣”。这句话放在书上也一样。（这就是所谓某种“观点角度最重要，是否一致无大碍”的观点）。我们可以尝试理解那些有趣人的观点——这能为你自己的思考提供情境背景。

这也是我为什么特意提到 Gregory Bateson 的原因。在接触 Gregory 之前，读过 Paul Watzlawick（编者注：出生于奥地利的美籍家庭治疗师、心理学家、传播理论学家与哲学家，是传播理论的领军人物）关于沟通理论方面的书，对我而言，Paul 的书总是更容易明白。而当开始读 Gregory 时一切却恰恰相反。《Steps to an Ecology of Mind》这本书我读过好几遍，现在我也会经常重读其中的某些章节。对我而言这本书常读常新，从不同的角度会读到得到不同的理解。

Alan：其实 Gregory 在该书中忽略了一些重要观点（我认为这部分是由于他的背景，部分由于其个性）。你可以在他的收官之作《Mind and Nature》中了解一二。当然，要更深入地了解“Mind”，更好的选择当然是读 Marvin Minsky 的《Society of Mind》及《The Emotion Machine》。另外还有较新的 Daniel Kahneman 的《Thinking: Fast and Slow》等。

Nick Lane 的《The Vital Question: Energy, Evolution, and the Origins of Complex Life》也不错，不仅谈论了精神（mind），也谈到了导致当下这种精神的因素。

Alan：这本书我倒不是太愿意读，如果一本书只是选题好或选题很重要，我不会盲目推荐。而我也对这类书不感冒。

那么你会不会推荐 António Damásio（编者注：南加州大学神经科学教授，著有《Self Comes to Mind》、《Descartes' Error》等）的书？

Alan：我不确定是否要将这本书推荐给 Anderson 顾问公司的人，这本书基调和情景都不错，但具体内容就一般般了。

是不是 Manuel Castells（编者注：西班牙社会学家，其研究引用数量，在社会科学领域列第五位）更具参考价值，尤其是《The Information Age》三部曲？

Alan：很棒的建议！在我列出阅读清单时就已经出版了，但我当时还没来得及阅读。可能如果带着特定目的阅读，内容过于冗长（也可能过于学术化）。另外，Mumford（编者注：美国历史学家，科学哲学家，以其对城市和城市建筑的研究闻名）也有《Technics and Civilization》三部曲，我倒希望 Anderson 顾问公司的人好好读读。当然 McLuhan 的系列著作对于大多数读者来说内涵更丰富。

Manuel 的书非常适于阅读，而且书中图表和数据的展示形式也为人称道。现在“三部曲”已经成为了通信、媒体科学领域的必读书。从更实际的角度看，《The Internet Galaxy: Reflections on the Internet, Business, and Society》值得推荐，比起“三部曲”也非常容易阅读，部头更小，而且解决了一些重要的问题。

Alan：这是《Gutenberg Galaxy》的姊妹篇吧，我刚订了一本。我会先入为主将它放在“不抱太大期望”书籍之列，但也值得一读。这种偏见其实有更大的来头（也是某种概括性的结论），虽然比较少见。通常都是不同的评论家（无论是科学界还是其他领域的哲学家）有了挺棒的想法。当然，许多所谓圈内人对周遭发生的一切还懵然无知。无论怎样，大多数好的评论都是来自行家，比如好的科学评论都是科学家写的，好的电脑评论都是技术人员写的，音乐评论也是如此，诸如此类。

你是否也会推荐：《The Laws of Form》（George Spencer Brown）、《Anything about Logic》（Gotthard Günther）、《Autopoiesis and Cognition: the Realization of the Living》（Maturana and Varela）等书籍？

Alan：当然。这里要强调一下，之前的推荐书单是给商界人士的——他们都很聪明，受过良好高等教育，只是阅读质量不高或“涉猎”不够广泛。他们目标都很明确，我想，他们上学并非是想吸取“普世知识”。当时我也尽量给他们挑一些效用明显的书籍。你这里提到的许多书都在《全球概览》（Whole Earth Catalog）中，它们也是帕罗奥托研究中心的最爱。尤其是《Laws of Form》。

那么 Richard Tarnas 的《The Passion of the Western Mind》、Jacob Bronowski 的《Ascent of Man》、Count Korzybski 的《Science and Sanity》、Vannevar Bush 的《Science is not Enough》、Shunryu Suzuki 的《What I Believe by Mark Booth (Ed)》、《Zen Mind, Beginners' Mind》，以及柏拉图的《Symposium》这些书呢？

Alan：人生确如白驹过隙！之前的书单是为商务人士准备的。因此没有“资本论”或其他的相关的政治书籍。换做我会把《Madison's Notes》放在制宪会议里，许多其他的政治文章受众也不一样。我那时没有推荐多少哲学入门读物（当然，这不是向他们推荐的重点）。现在我想起了原始版本的清单（当时是以电邮形式发送的），不仅仅是因为我不知道网络上的图画形式而得以由此拓展，也因此而进行了修改。其中最重要的改变包括了一些句子，如“除了 Betrand Rrussell ”以外的人或 Jerome Bruner 之外的书目再到一些零散的书目（如下一层级的书目）。但是，关于柏拉图，我说了什么倒想不起来了。

《The Dream Machine》这本书出版了 15 年了，你认为对当今这个“大数据”和“机器学习”的时代，是否有帮助？

Alan：这是迄今为止关于 60 年代研究基金资助和社区最好的书了。我们这个时代或任何一个时代都能因这本书而得到更充分的滋养。例如，真正亟待解决的不是“大数据”，而是“大智慧”，不是“机器学习”，而是“机器思考”。而许多投资者愿意花大笔金钱去研究的所谓“梦想机器”实际上是“寻找问题”，而非“解决问题”的机器——今天要做的还有许多。

你未来会给这份书单来个升级版吗？

Alan：有许多其他清单书目需要慢慢完成。最基本的作用在于它们能带来“多重角度”，以及并非所有的思维角度在种种情境之下都能同样有效。最初我提供的书目是给“商务人士”。儿童则需要其他书目，不同的人有不同需求。

阅读很重要的一个特色就是可以产生愉悦感，尤其是当你

见到一本恨不得马上就读的书。曾经，我每年都读好几百本书，直到我开始念高中，我不喜欢老师为我们挑选的书目。我觉得，与其让学生就特定“主题”阅读老师推荐的书籍，不如让学生自己去图书馆浩瀚的馆藏中自行挑选。这能让学生更好地就特定主题进行趣味性探讨。但最终也没落实，所以我放弃了，于是像班上的其他学生一样，读起了所谓的“经典喜剧”（后来我的确抽出时间来阅读那些“挑选书目”）。

Mortimer Adler 曾编选“西方世界伟大名著”（The Great Books of the Western World，共 60 本）。除这些书外，他还挑选了 102 本所谓的“伟大思想凝结成的著作”，每本书他都写了篇书评，并且与之前的“伟大著作”系列进行了组配索引，后来将这些论文发表在了两卷《Syntopicon》中。

从这一意义来说，我为 Anderson 咨询公司提供的清单书目就太短了。所谓的“伟大著作”也是如此：书目也太短了。另外一种办法是所谓的“津桥”法（Oxbridge approach）——挑选“四个”最重要的尽量宽泛的主题，进一步向内推敲，直到遇到真正有价值的“干货”。

“津桥”之道的一部分就是有位“阅读推荐师”，专门负责帮你挑选阅读书目，对你来说，这样的人是无比重要的财富。

归结成一句话：广泛阅读。

你曾说自己最喜欢的电子游戏是 Rocky's Boots，对于那些学习编程的人，你有没有游戏可以推荐？

Alan：我可能跟不上电子游戏的潮流了，所以给不出好答复。过去我喜欢的 Rocky’s Boots，因为游戏的内容和理念都挺新奇有趣——两者配合得天衣无缝。我也挺喜欢这个游戏的后续版本 Robot Odyssey，后来我建议 TLC 公司的同事用类似商标而非洛基的电路图来表现机器人语言（后者与表现需求不太吻合）。我的确尝试让信奉马克思主义的人将 Sim City 打造成一个以规则为依托的体系，孩子们也可以加入编程，这样他们就能更好地了解发电机原理，并且进行改造（只能靠努力，不靠运气）。

今天，如果你环顾四周，寻求对孩子真正有益的深层次内容，可能 Cellular Automata 会是排名前十位的选择之一，也能帮助他们更好地参与一些伟大的游戏。

你推荐的书籍当中，哪本更重要？

Alan：这有点像拿伟大的作曲家做比较——比如，要判定是巴赫伟大还是贝多芬或其他作曲家伟大。我认为更好的方法是设定一个基准：在基准之上就能被称为“伟大”。但要订立这样的基准并不容易。

在这张书目中，我并没有囊括太多我现在视为“伟大”的书籍——我把他们放进了可阅清单，针对特定受众。仅以这本为例吧：牛顿的《自然哲学的数学原理》是一本经典之作。读起来有些晦涩难懂，但对于这群读者来说是挺好的选择，无论用哪个标准衡量，都是旷世之作。

你通常会读纸质书籍，还是也会在平板电脑、手机上阅读？

Alan：我在帕罗奥托研究中心做了许多关于可读字体的设计，几年后，也开始研究从各种 CRT 上阅读。当时普遍反映从各种显示器上阅读会带来所谓“不好的感觉”，所以才引发了类似的研究。这些显示器今天已经好很多了，但我还是会有这种“不好的感觉”（可能较之前已有改善）。这是由于眼球的移动和人脑记忆落差所致，所以我尽量将阅读内容打印出来再读。

毫无疑问，电子媒体的可读性跟纸媒一样——另外，从某种意义上来说，如果可读性不如纸媒，也没什么好惊讶的（我想大多数技术专家都不会在意）。我很好奇为何对超高清分辨率的屏幕的研发不能精益求精，从而提高可读性。

第一台商用激光打印机达到了 300dpi，效果非常好（其实始祖还另有一台——由 Gary Starkweather 在 Parc 所设计的 500dpi）。我问过 John Warnock 为什么 300dpi 比想象中效果更佳，他说那是“纯黑”效果，精准性一流。

我们现在至少拥有了某种“视网膜”级别的显示器，但我想还有一些闪烁的光点会造成阅读问题（若真如此，这将驳倒很久之前那些认为只要显示器达到 120p 以上就不会造成视觉困扰的实验了）。

但如果我用笔记本，双眼还是需要加大扫视的力度，电子版的对比度和分辨率都不太好。

我在帕罗奥托研究中心所做的实验，与 Tom Comsweet 在 SRI 的工作结合一起，能表明对比度的确不稳定，离真正的纯黑显示还有相当长的距离。

人为因素也糟糕地影响了设计，例如 iPad 和 E Ink 设备不能在手中取得良好的平衡。希望下一代类似设备能减轻重量。

如果显示屏安装了“四分之一波片”可以抵消许多反射光——我不理解为什么最近的光面显示器再度回归（是有更好的光线效果，还是更廉价？）我用的是一台老旧的 Macbook Air，没有高度反光的屏幕。黑色显示相当不错。我觉得使用时眼睛跳跃度稍大，因此（或由于其他原因），这也会让我很明显地感觉到“我记不太清了”。这可能是个幻觉，当我尝试想起过去某个节点读到什么的时候，可能要重新去找一遍。

有观点认为看电子屏幕阅读时，人们往往只专注于静态细节，而非流动的信息，你怎么看？

Alan：这一点很有趣（让我想到了 Steve Wyer 在 1980 年的一篇论文）。里面有一个讽刺挺有趣，就是在所谓强调清晰精准的媒介里展示了一个抽象的结果。如果他们尝试了包括阅读流畅度的分级评分就好了。

百问 Alan Kay

译 / 郭韵

文化与教育

你回顾计算机在 70 年代和今天对比时，总是着眼于它在教育方面发挥的价值，然而，计算机如今却耗费了人的大量时间和精力，这方面你怎么看？

Alan：那时我们常常思索如何令人类更先进，Douglas Engelbart 和他的团队则干脆用“人类增强”（Human Augmentation）这样的字眼来描述这些探索，其中包括教育，还有其他林林总总的事物。从 1965 年摩尔定律被提出到十年后电视的盛行，如果没有摩尔定律，这些“精神鸦片”不可能被广

泛传播（我曾经说，电视是人类发明的最应该加上类似“医师警告”标签的东西）。其实在电视兴起之前，我们就已经有了精神鸦片，想想早期的电子游戏《星球大战》多吸引人。这是工业文明蓬勃发展的一部分，“工业”的标志是它产生了“剩余”，而“文明”的标志则是教导孩子如何在这个过度富裕的世界里避开物欲至上的陷阱。在美国这样一个只关心赚钱而不是钱从哪赚的国家里，这是个严峻的问题。

关于电子产品的使用，过程似乎有点像自然界中的“选择法则”——善于选择最适合自己工具的人，更容易生存。如今我们没有办法避免儿童使用电子产品，比如网聊、网购，甚至赚钱，但这一过程消耗了他们的大量时间，同时对身心也造成了一定的影响。你有哪些切实可行的办法让这一现状不要继续恶劣下去？

Alan：我有位朋友叫 Neil Postman（一位很棒的媒体评论员）提倡要教育孩子成为“游击队战士”，以此抗衡他们所面临的成千上万靠希望侵蚀掌控他们大脑来获益的机构。大多数孩子——包括大多数父母——都没意识到这些铺天盖地的信息不仅来势汹汹，而且弊大于利。

Neil 认为，所有人都应该意识到自己所生活的环境，以及大脑怎样以一种我们难以察觉的方式对这一环境进行处理和适应。遗憾的是，我们往往意识不到自己到底接纳和吸收了什么，就形成了一种“新常态”。

换句话说，这一过程就像科学研究的起点所描述的：世界并非如它看上去那样。“作为人类，我身上集结了各种特征和行为，许多特征具有返祖性，甚至妨碍了进化”。想想人们曾经一度对盐、脂肪、糖、咖啡因等物质非常渴求，结果食品公司开始在食物里添加越来越多这类物质。另外，Neil 也指出，像《娱乐至死》（Amusing Ourselves To Death）和《童年终结》（The End Of Childhood）一类书里所描述的，我们对新闻、新鲜事物、刺激一类事情非常迷恋，所以消费品公司不断让各种信息发放渠道充斥着这些事情。

许多这些想法都可以回溯到 Marshall McLuhan（译者注：20 世纪原创媒介理论家、思想家）、Harold Innis（译者注：多伦多传播学派创始人之一）时期。但底线是，孩子们必须学会如何利用二十一世纪，否则他们很可能失去二十一世纪。

有时我们会对年轻人感到厌烦，就像厌烦 Node.js 一样，年轻人对他们所做的事怀有热心，但他们好像总是要“改变世界”，搞一些“发明创造”，在一起没日没夜地讨论。你认为除此之外，想要激发创造力还有其他更好的方法吗？

Alan：过去人人常常被告诫不要去“炒冷饭”。现在我们生活的时代似乎花了很多时间“新瓶装旧酒”。原因是这些装旧酒的人跟原来的酿酒师不在同一个频率。这是现在所谓“流行文化”的症状，即“身份认同”和“参与”本身比实质性进步更受人重视。

那怎样才能跳出这种所谓的“流行文化”呢？

Alan：我认为这首先是一个认知的问题。在美国，我们都陷入了一种流行文化，这种文化是如此甚嚣尘上，撼动了所谓的“成熟文化”。这种强大的渗透性让它独自为大，一叶障目，也让那些关心别人思想的人很难不去过度重视流行文化的各种规范。其二，不均衡也是造成这一问题的原因，成熟的艺术一直都需要流行艺术来对比衬托，以此获得模糊的“身份”，而盲目跟风造成了人们对流行文化的反感。就像盐本身是很好的调料，但要做一块蛋糕，仅仅有盐是不够的。

关于这一点，我从 McLuhan 的作品中受益匪浅——尤其是他关于媒体以及媒体所形成的环境的观点。60 年代，我又开始探究人类学（在这个时代之前，人类学被过度政治化了）。现在，如 Khaneman、Thaler、Ariely 等“行为经济学家”的书都很不错，深入研究了人们在各自环境中的实质行为。

另一个看待此事的角度就是找到不同的方式来“真正的地获得知识”，如此一来，才能让本土的变成全球的，让单一的变成多元的，让教条变成多重角度的观点，让即兴创作变成流传的作品。总结一下就是，最初的发明不会失去用处，只是不会一家独大。

那么在你看来，一个群体甚至一个文明想要冲破这一桎梏该怎么做，当我们面临选择或者处于混乱的状态，知识是否依然那么有用？

Alan：所有文化都包含了无数知识——其中更大的影响是情境以及认知带来的影响（比如观点角度、立场，以及价值观等）。另外，自我觉知，意识到我们是什么样的人是重要的一步，这也是教育的真正意义所在。这意味着我们要抽离于所有人类的偏见之外。

编程如今已经成为了一种“流行文化”，你怎么看？

Alan：我不认为“流行文化”的种种方式是完成大多数事情的最佳方法（虽然“时不时”这些方法也偶有建树）。 真正的问题是“一次骇入是否能重置所谓‘常态’的概念”（does a hack reset “normal”）？ 对大多数人而言，的确能发挥这个作用。这也让他们很难顾及到实质方面。

几年前我曾经说过一句话“追求更好更完美是追求现实需要的敌人”。今天计算机从业者中很多人犯的一个大错就是没有真正关注现实所需。 他们也没有从这些需要出发。实际上，这就是为什么人类总是追求进步，并且会说“至少已经变得更好了”。但如果没有满足基本需求，就不可能向“更好”迈进，只会徒劳无功自欺欺人。

我有位朋友是高中教师，他说在教育中科学代表的仅是一些课件、课程计划，但这些东西在实际的教育中只占很小一部分，而师生之间的交流更为重要。有哪些工具可以让这一过程更丰富多彩？

Alan：很难推荐。因为万维网时代的人并不懂得互联网或计算机媒体的内涵，却要基于他们的浅显理解，去希望一些并非“所思即所得”的拙劣想法能发挥管中窥豹的效用。

我认为，了解传统的口传文化和思考之间的“真正良好”的平衡很重要，读写能力，尤其是通过大众传媒带来的知识素养，能给相关人士带来什么，计算机和无处不在的网络作为真正有用的发明能带来什么。在某种意义上，真正的知识素养正回退到沟通的“口头模式”以及思想的“口头模式上”——例如，“发信息”实质上就是口头表达转述，而并不是某种文学形式。

这真是一场灾难。

然而，即使是自学的人也需要一些口头交流，问题在于如

何平衡。阅读流畅的人获得信息要比口头传递快得多，手头也更能获得更多不同资源。这意味着在二十一世纪，大多数人应该读更多，尤其是学生（阅读的时间应该远远多于谈天）。具备责任心的成年人，尤其是教师和父母，应该极力促成下一代拥有更多的阅读体验。

最后一点，我建议这位朋友仔细读读 Daneil Kahneman 的《Thinking: Fast and Slow》，这本书里谈到了各种实际互动之间的利弊权衡，无论是与人，还是与计算机之间。我想大多数人长大的过程中，都会错过成为真正会思考的人的机会，因为他们长大的环境并没有反映出这些相关要素和锻炼他们如何权衡的契机。

有个问题一直困扰我，为什么你总是觉得那些“知道怎样做事”和“用最好／最快／最经济的方式把事情做好”的人之间存在很大的差距？

Alan：我在 PARC 工作非常幸运，既有资金资助，也有时间充分思考，甚至犯下错误，虽然外人未必知道。你提出的确是个普遍问题，这让我想到一个例子。猎人和采集者（代表我们的遗传属性）找到土地肥沃的山谷，进行大肆采掘，然后打一枪换一个地方（当然都是小规模作业）。“文明”则是部分关于学会通过教育和规划，克服危险的返祖现象。普遍而言，这正是我们应该做的。

我想问几个“大”一点的问题：你认为我们怎样才能让今天的世界变得更好（不仅限于科技手段）？人类这一种群的未来将走向何方？人类文明怎样才能升级为更高级的文明？尤其是怎样消除贫穷、痛苦方面，当然前提不是摧毁地球，而是通过政治、教育以及社会手段。

Alan：“凡人是多么的愚蠢啊！”Puck（莎士比亚名著《仲夏夜之梦》中的精灵）的意思是我们都很容易被愚弄。实际上我们喜欢被愚弄，甚至情愿花钱去上当！

任何人都可以了解一些最重要的想法，比如去了解“人类似乎从火星而来”的观念。这意味着我们开始超越自身，并开始与我们的天性作斗争，去了解那些危险并可能产生与预期相悖的事物。这跟大多数学校自以为是的自我定位非常不一样。然而，在 60 年代，伟大的 Jerome Bruner（译者注：美国教育心理学家，认知心理学家）为五年级的学生制定了一个很棒的课程规划，为 K–5 教育中“真正的人类学”订立了一个绝佳开端。

这指的是 MACOS 项目吗？

Alan：是的，当时 MACOS 关于这方面的推荐书目是《课室里的政治》。网上有许多关于 MACOS 的资料。Bruner 写了许多很不错的论文，都构成了 MACOS 项目的一部分。

对于正在学习计算机科学的高校学生你有哪些建议？

Alan：多学知识吧，至少学习一门真正的科学，以及一门是真正的工程学。这些知识能够帮我们去校正所谓的“计算机科学”中的奇谈怪论。此外，我还建议多学点人类学课程，以及各种社会心理学等。我也建议有资源的高校可以开设“媒体理论”，或类似“Mcluhan”，“Innis”，“Postman”类型的课程。

我们该怎样教育下一代编程知识呢？是使用电脑作为介质还是像树莓派那样的开发版？

Alan：是时候发明一种属于儿童的编程语言了。几年前我希望做的版本会是“待续：Etoys 儿童软件”，其中一部分应该是：儿童素描（Scratch）（由同一群人的一部分抽调出来做，但只是一个附属部分）。

你认为编程要作为一门学科在学校里普及吗？我担心如果这样做，很多学生会像今天讨厌数学一样讨厌编程。

Alan：每个人都应该先学习“真正的科学”，许多其他的知识也可以由此顺势发展——包括能够真正帮助大多数人的不同编程方法。

你如何看待人类的思维？

Alan：Bob Barton 一度将程序员称为“低劣邪教里的高级祭司”，并指出“计算应该归入到宗教学”（ca 1966）。

思维无论从哪个角度来看都是艰难之举，其实人类并不太擅长此事——从基因而言，我们是“通过记忆习得来学习”（而不是从真正理解来学习），我们也是“不断回想来进行思考”，而非真正通过实际推理来思考。我们需要面临更高维度的事情就是在 Kaheneman 的“系统”里面的各种缓存在缺乏实际思维的情境下，提供各种实时反应的支持。

你怎么看“数字安息日”主张，以及与之相关的论调，例如：Engelbart 让人变得更智慧；关于鸡生蛋蛋生鸡的问题，网络给你带来新的思考方式了吗；以及信息过载等。

Alan：蜡烛、美酒和面包就不代表技术吗？很难相信这种说法。（我喜欢弹奏音乐，音乐和乐器都是技术的代表）。不要陷入那些毫无营养的所谓的“合法药物”。

“我们已经变得愚笨”——这就是为什么原本可以变得更好的世界还是原地踏步。我们必须从自身找原因，一个积极的解决之道就是去问问“真正的教育能够帮人类解决什么问题？”

在一次演讲中你提到，思想是不需要逻辑的，这一点是希腊人没看到的。

Alan：这就是将“科学”与“数学”混淆了（他们都这样做了，像我们一样，他们都深受其中的整齐所影响，并且深深地了解“理性思维”能带来多大好处），而“科学”正是他们所缺乏的。“数学”是不受外界影响的，科学正是我们的各个表征系统与“那个东西就是什么”之间的一种博弈较量。

我发现你的作品与 Christopher Alexander 的作品在形式上有些相似，你怎么看？

Alan：迄今，我最爱的 Alexander 的著作是《关于形式合成的笔记》，但他却否认了这个作品。

如果你为天赋极高的高中生设计课程，你会怎么做？

Alan：我觉得学生都可以跳级（early grades），尤其是天赋异禀的学生。其实他们的认知在很早之时就成形了——虽然不是根深蒂固，但却很难改变——而这些学生所学到的知识正是我们应该集中资源与努力去巩固的。

在你的不同关注点和爱好中，你如何游刃有余？

Alan：先从大的说，如果将“不同系统”作为一种通用语以及维度坐标来思考宇宙、世界万物、不同文化，以及社会系统，我们会怎样看我们现有的各种技术，以及我们身处的各种制度呢？这是让 21 世纪儿童开始着手努力的很好的方向。

你那么关心教育，又对今天的教育现状颇有微词，那么你认为把现在的技术带进教育中是不是个好主意，比如

用游戏的方式帮助儿童学习，或者在 UI 上做些改善，比如减少或隐藏一些多任务处理的界面等。

Alan：这里有几个问题。UI 设计的一个重要原则就是要站在用户的角度，这也是教育中的一个重要原则。正如 David Ausubel 所说：所有的学习过程都是在已知的边缘产生。对于儿童（包括很多成人），他们思考、学习和领悟的过程是通过故事来驱动的。然而，很多重要领域，例如科学或计算机的思想却不是以故事的形式呈现，所以很多现代的学习方法，是帮助学习者从这种人类的本能中走出来，为他们提供多种可能的学习方法。

还有——我们所沉浸的每样东西都会让“常态”被颠覆，对于很多人而言，这种现象的发生是不知不觉的。这应该成为“辅助学习”过程中一个很具备觉知意识的一环。

对于计算机专业毕业生，你会建议他们去很火并且有很多优秀人才的领域，比如人工智能，还是建议去一个相对冷门但有空间可能大有作为的领域？

Alan：请尽量避免给未来太多规划，只需要知道事情具体进展到哪一步。任何好的教育经验都会让那些寻觅者找到目标，而非让他们误入歧途。请注意，教育不够的人不适宜做太大的选择，所以尽量学更多的东西吧！

关于创新中流行文化与进程的论点，可以多讲讲吗？

Alan：尝试去了解“流行文化”的本质，尤其是当它与“传统文化”或与人类基因学相连时。 比如流行文化需要什么呢？人们真正希望从流行文化中得到什么？以及为什么要这样做？成熟文化的涵义是什么？其中利弊各是什么？在不同文化中，分别包括着什么批判观点和裨益？

语言、工具与程序设计

在《The Power of the Context》中你写道：编程中第一“不该”就是“不该自己写工具”，包括语言、操作系统等。对于崇尚 DIY 精神的人，DIY 的过程会令其技术长进，而另一方面如果写工具的过程过于复杂，又有点得不偿失。在你看来，程序员什么时候应该使用现成的工具，什么时候该 DIY ？

Alan：这个问题可真问住我了，每个年代都有一些项目，相关工具的开发让整个项目整体进展陷入泥潭。在 PARC 能与 Chuck Thacker（译者注：因设计与实现了第一台现代个人电脑 Xerox Alto 而荣获 2009 年图灵奖）和 Dan Ingalls（译者注：Smalltalk 语言设计者之一）等真正天才共事让我获益匪浅。其实让工作进展到第二阶段与前功尽弃之间只是一步之差。

思考这个问题的另一角度就是不要“模式化”，如果的确有相当不错的独立模块系统，甚至也有硬件在支持这一系统能让不同团队的人一起工作，而不至于陷入“打各种补丁”的泥潭，倒也不见得是坏事。在 PARC 工作的日子里，我最享受的事情之一就是看 Dan Ingalls 怎样神乎其神地基于旧系统改造出新系统，尤其是新系统还能更好地促成下一个新系统的建立。

我并不太喜欢用 UNIX，因其平台不能及时反映出其中各种想法，但如果你充分考虑其文化历史发展过程，其中几个努力方向挺值得称道——包括使用了小内核，可以使用 UNIX 进程进行所有系统构建（这是一个非常有用的“OOP”版本——由于其执行进程的特点，操作对象不能太小）。但看着这个非常不错的组合渐渐转变成了各种沉重的负载和依赖，也的确挺沮丧。其中一个原因是因为将一些非指令的信息解析到每一个进程中——在 PARC，Smaltalk 1.0 就用到了这一方法，因为各个信息构架缺乏少与一种真正语言的连接（HTTP 也遇到过类似的事情，如果有人注意到这一点，后续发展想必会有所不同）。

Smalltalk 是一种将非指令信息传达给对象的早期尝试，蕴含的哲理是，你如果留意了组成这些信息的各种规例，就能获得某种“编程语言”。

那么 Smalltalk 有哪些不足，或者不够与时俱进之处？

Alan：我们要区分什么能升级，什么不可以。例如，name 基本作用于局部，我们本可以期望找到更好的方式去描述各种资源，或者“传递的是各种过程，而非信息”。想想关于 PARC 使用什么来生成 PostScript 的方法，而不是直接为打印机尝试定义某种文件格式或生成不同文档。

但如果通过各种变量与选择器来指代事物，这些名字必须在局部被解释。Smalltalk 也能通过描述找到许多不同事物（例如：它能通过输入和输出数据来找到许多“正弦函数”）。

当今没有其他编程语言能像 Smalltalk 那样自我升级，很悲哀，不是吗？

Alan：这应该归功于 Dan Ingalls。我会去“观察”可能性，但 Dan 能很清晰地在应该做的事情和能够允许我们在 70 年代在相对小型的机器里取得进展之间，做出非常重要的取舍。这里要重点感谢 Thacker，他在 PARC HW 研究中展示了“可能性的艺术”。

我过去很喜欢 MOP 这本手册，因为能将模型更深入地与实际定义结合起来——我很喜欢将“设计和执行空间”作为整体，而非点到点的局部改善。

对于学习 OOP，哪种语言最有帮助？

Alan：好问题。如果我们谈论的是“真正的 OOP”，那么现在发展如何我就不大肯定了。不知其他人怎么想？ Smalltalk 已成为今天屡屡谈及的话题，它依旧在 oo 方面表现出色。Erlang 还有它的衍生语言也很有趣，也能帮助思维。

你提到不该重复造轮子，你是否认为工程师就应该做个能善用工具的人？

Alan：工程学很了不起——但请想想真正的科学发明以后，出现了怎样的境况！今天的“计算机科学”一方面更像“图书馆科学”，另一方面又跟工程学更为一致（其中通常都体现了高度的工程艺术）。如果我们能用软件和大多数设计来做真正的工程和科学，就再好不过了。

近年来，像 Haskell 这种通用的纯函数式编程语言开始流行，不知你是否有所关注。但在硬件产品的研发上，我们似乎还是纠结于动态可变的语言和功能静态语言之间，你对此有什么看法？

Alan：先不说 Haskell，我想开启“各种进程的最优模型”更为重要，并且让这些模型与我们在原先的模型所衍生的各种语言的硬件最佳操作方法实现契合。我不认为各种编程语言造成的“泥潭”与人类所面临的其他“泥潭”类似，后者是如此贴近人类，很难让人从中抽离出来去看待其他可能性。

你认为对于程序员来说，发明一门语言是必要的吗？还是由那些系统级的开发者发明，其他人用就好？

Alan：这个问题有难度。要回答得先问“计算机科学”和（比如）物理之间会有内在的联系吗？抑或是计算机只是科学发展的产物？

可对于物理学来说，进步很容易看出来，比如你可以解释更多自然现象。但对于计算机，它把很多主题揉到了一起，这一点如何证明？

Alan：好问题，我试着解答，但不确定一次能说得明白。毫无疑问，我们不得不收回“计算机科学”这个词语，并尝试根据构成真正科学的要素给予它真正的涵义。Herb Simon 指出，这是“人工科学”，意味着它是一项关于我们能研制出什么，已经研制出什么的研究。

科学尝试通过制作各种模型并对效能进行评估来理解现象。大自然孕育各种现象，工程师也可以，比如利用他们的知识建造桥梁。如早期工程学中的大多数事物一样，他们把关于桥梁的各种知识像菜谱一样写下来。有了科学之后，科学家和工程师得以使用现存的各式桥梁作为研究对象，并发明关于桥梁的各种模型 / 理论。这在最近变得尤为流行（我出生后几个月，塔科马海峡吊桥就坍塌了）。

第一届图灵奖得主 Al Perlis 在 60 年代被问到“什么是计算机科学？”时说：“计算机科学就是各种进程的科学！”“各种进程”包括计算机科学，也包括生物学、社会学等等。他的意思是，计算能为几乎世间一切建立更好的模型，尤其是各种动态事物（任何事物都是动态的）。

今天，我们也能为“让‘计算机科学’重回它一度被遗弃的原点”出一分力。

在任何情况下，这种观点与工程学非常不同。要知道，“人工科学”吸引人的地方，就是你必须通过各种现象和模型来塑造人工制品。

我在准备一个关于编程语言对思维模型影响的资料，可以告诉我哪些编程语言跟你的思维模式更相近吗？

Alan：对这个问题我会问，“编程语言中的哪些设计能帮助我们思考？”我们现在的思考层级太低了！在如今这个时代，缺乏安全的元定义（meta-definition）令人震惊。

能具体谈谈元定义吗？

“元”是危险的，语言内的安全元语言会通过“各种屏障”进行保护。请注意，一个变量负载的“任务”就是功能性语言中的“元维度”（可能会在必要时为提高安全性使用某种“回退到‘世界的机制（如各种交易）’”）。与不同种类的优化类似（在某种意义上违反了模型界限）——我们有不同途径让安全性得到提高（大多数语言都没起到太大作用）。

Aspects 跟 MOP 是一回事吗？

Alan：我不认为 Aspects 跟 MOP 可同日而语。但其中的“超连接性”正是语言和开发系统都应该大幅提升的原因。例如，Dan Ingall 在 Smalltalks 上付诸了许多努力，让它能安全地 Debug，即使是在非常深层的架构里。即使是他在进行那些突破之时，我们都应该知道还有许多更深的层次值得被挖掘。也可以想想另一个例子，Goldstein 和 Bobrow 在 Smalltalk 中应用了 PIE 系统，这也是我最喜欢的元系统之一。

我记得 80 年代的 Saber-C 还允许在未进入重启进程时热编辑，是否如今的编程工具不如从前？

Alan：在 60 年代，你可以“热编辑”（hot-edit）Lisp 语言（BBN 1.85）（也有其他系统）。在 70 年代 PARC 的 Smalltalk 语言里使用了许多这些想法，并且基于此有了进一步发展。

编译器能否在开发时就基本覆盖所有目标系统？我知道你一向不看好 HTML，对于 HTML 你有建设性意见吗？

Alan：TCP/IP“在开发时就能基本覆盖所有目标系统”，这一点有道理但不全对，因为它是早期创作，部分由于其简洁易懂，也由于它并没有在实质的信息上定义各种架构，只是在“envelope”上定义简单的架构。这也部分由于“/”符号不能构成单独的理论。此外，在它之前并影响了它的 PARC PUP“互联网”——都是尝试构建架构的例子，因此各种模型才能基于对双方的最简假设进行互动。

下一步，为各个内在含义组织最简基础（不仅仅是互联网），在 70 年代通信系统的各种想法得到长足发展之时已经经过了充分酝酿，但当时还未切中要点，而且成熟度也尚未在 1983 年的“国旗日”成为整体装置的一部分。

然而，我们还需要思考，能部分构成 TCP/IP 的最简元素是什么？让“文字的意思”而非字节能被发送，我们在接收端需要作出哪些假设来保证一个被传送“意思”的安全？

现在再考虑这些会不会太迟了？

Alan：我认为不会。但这会要求整个计算机界对计算的普遍认知都进行大幅度调整，比如迭代、看法和不同的目标。

你怎么看当今的计算机语言的设计，哪些语言你觉得设计的不错，哪些你觉得不太好？

Alan：我觉得都设计得不咋地，看着就生气。用 21 世纪的眼光看，语法都经不起推敲。但如果从一些过去被发明出来的语言角度来看，小部分语言还挺有趣，例如 Erlang。

这个问题是关于人机交互的，我们知道人类的沟通需要互动，而计算机语言的数量远远未达到人类语言的数量，而现在我们可以通过网络轻松找到需要的服务，这样一来，沟通是否依然重要？

Alan：请注意，沟通最早可能源于某种向神灵祷告的行为，从而使人超越某种身体极限。我们完全能用普通语言来表达数学思想，但与此相反，我们尝试让一些数学概念不再那么模棱两可，就必须去学习不同的惯例规矩。既然如此，你有没有想过（或提到过）我们需要互相沟通才能理解某些意思呢？

程序员：你曾认为 LISP 是最好的编程语言，在近年的 Haskell 和 Scala 出现之后，你还这么认为吗？

Alan：我要澄清一下。我的确切意思不是指某种可编程的语言，而是指作为：1. 某个“构建材料”；2. 尤其是“可以思考的某种手工作品”。一旦你通过感觉捕捉到了其含义，大多数编程语言里的难题（包括今天的难题）都变得更为可控了，可批判性也更强了。

你是否觉得使用 Smalltalk 语言编程依然具有其优势，抑或是它的某些优势在其他编程语言里也能找到？

Alan：Smalltalk 在 70 年代是那个时代的“伟大发明”。其中一个很实际的例子是，能实时快速使用多元媒体与不同互动形式来运行，并让其升级到 64k 字节（或者更多）。PARC 最初就没有打算让它在很多维度上可以迭代。

我们必须要想想为什么这个语言在今天更值得一提。

你认为用 Meatadata 编程的下一步应该是什么？

Alan：我们领域中最大的盲点之一就是集成升级问题（具体为何我也不太清楚）。这导致了几十年前同等级别很棒的想法需要花费巨大努力才能企及。可能之前我在分子生物学的背景帮助我对这一问题看得更透。在某种意义上，想要完全理解这一问题，需要我们了解现在的最新发展，以及这些发展为什么能发挥如此良好的作用。

实际上，如果我们将集成升级的不同维度都考虑在内，编程语言会展现出什么姿态呢？

那你是否认为 Actor 是个不错的可升级模型？

Alan：我非常喜爱 Actor，但它的可升级性能到底如何还值得商榷。这里我列举一个简单的类比：假设有一个很棒的原子模型，其中在多大程度上能作为一个好途径用在现有的不同系统之上？或者我们是否能想出一些更好、更有用的架构能更好地契合我们目前想解决的集成升级问题。提醒一下：往生物学方向想，不用考虑原子物理和化学。

这里需要指出，物理学的发展对这一问题只起到了部分作用。因此，让大多数早期语言“变得更好”可能会错过大部分互联网或更广泛空间内所需要的真正进展。我个人认为，大多数企业软件即使到了今天也需要架构改良，在种类上要有别于 60 与 70 年代的语言。

那么，我们该如何才能达到 scale-free ？

Alan：尝试使用 UNIX 作为“模型”或“Objects”的众多问题之一就是它们包含了不那么“Objects”性的东西，如字符串等，这就让后续的不同集成升级或使用拓展变得尤为艰难了。

这实际上并非太 scale-free，但我之前在别处也提到过要“找那些不太好干的活，把它干得漂亮”。接下来我们可以看看如何 scale down（要完成顺利的 scale up 几乎是不可能的）。这就是 Smalltalk 的原理——我想到过可以“干得漂亮”的 20 个例子，其中一些是“规模较大”（在那个特定时代而言）。我们（尤其是 Dan Ingalls 和 Ted Kaehler）都能寻找到不同的方式令更为广泛性的事物更具体而高效，从而在我们必须面对的不同架构级别上能做到统一协同运作。

在这次访谈的其他部分，我已经在谈及与世界的关联时提到了一些问题（想想“星系的”这个词来帮助我们思维！）今天的各种语言或操作系统里没有任何关于“生物学”现状元素。

另一方面，有几个开端都省略了通过信息传递的编程到信息接收的编程这个步骤（总体而言也挺棘手的），尤其是通过意图 / 含义协商（intent/meaning negotiation）来进行编程，诸如此类。

Linda 是 80 年代的一个极好想法，类似的想法如果升级到 40 年之后使用，会变得怎样呢？显然不会再像 Linda 一样，所以不要以那点为思维原点

你怎么看像 Haskell、OCaml 这种函数式编程语言？

Alan：这些语言需要更好的时间机制（如研究 McCarthy's fluents 的方法）。在此基础上，我们还要考虑这些语言是否能开发系统级应用。我喜欢对函数的定义明晰化，这个想法也可以继续推广，但我想可以用不同的方法来运用。可以去看看 Bob Balzer 在 60 年代末和 70 年代初的 EXDAMS 系统。

你认为消息是传送给可以对它作出反应的接收器的一种指令信息吗？

Alan：我喜欢“信息”中的“非指令”元素。我在 Smalltalk 第一版也留下了一些成果，因为我觉得 Simula 版本语义性不够强，现在我也还是这样认为。

我很欣赏 Smalltalk 的强大与简洁，你是否可以给其他语言设计者一些建议？

Alan：你想让未来的程序员学习哪些东西？对你而言，你最熟知的思想能决定你的认知负载。如果用音乐来打比方，就像从卡祖笛到大提琴中的音域中寻找一个答案。语言中不应该含有的就是“种种无谓的困难”（或无谓的犯傻）。这里的意思是说，值得去想想我们引入到自身的文化中的种种问题了，学个自行车都有所谓的学习曲线，更不用说飞机了。或者，在真正的物理学中，张量微积分处于什么地位呢？

你认为未来人类跟机器交流有可能吗？现在人们按照自己的方式为机器植入语言能力，但似乎并不太成功。

Alan：在工程学中，其中一个最有趣的过程就是建立模型来圈定你所建立的最终目标。听起来有点奇怪，但正是这个模型成为了“所尝试的意图”的集中体现。

现在，我们可以沉思，在允许模型建立的时候大多数语言有多粗劣，并且不同的优化提升都是各自为政，而非融会贯通的。

在之前的采访中，你表示比较看好仿真技术，现在也一样看好吗？你认为仿真技术是计算机革命的一个有力保障吗？至于下一步该怎么走，你有好想法吗？

Alan：是的，我依然看好仿真技术。至于下一步我没有什么好想法。近几年也出现了一个有趣的选择——即 NetLogo（从 StarLogo 衍生而来）。对于各种人群都适用。另外，StarLogo Nova 同样也值得去了解。

但我想强调一句，这些并不是我们在 2016 年及以后所需要的真正的通用模拟语言。

如果用 VHDL/Verilog/SystemVerilog 这类语言开发（当然涉及到仿真和时间的概念），这跟你提到的仿真是同一回事吗？

Alan：那我先反问你：如果模拟一场瘟疫，情景会是怎样的？或者模拟蚂蚁通过各个信息素去寻找食物？又或者模拟染料在水里徐徐扩散的情境？模拟物体自然下落呢？

你认为 WebAssembly 现在的发展方向正确吗？当然，我不是说要它取代 HTML。

Alan：这是“系统的盲目性”在面临真正的升级问题的绝佳例子。浏览器在进行升级之前需要了解什么？为什么浏览器开发者们认识不到这一点呢？要知道，这些知识在 90 年代初期已经很流行了。

你怎么看 Self 语言，我曾经读过 STEPS Toward The Reinvention of Programming，觉得两者关联挺大。

Alan：我挺喜欢 Self 语言。“Good OOP”仍然在等待一个

更好的理念更新来取代“Class”。

找到正确的编程语言有多重要？

Alan：要处理一些更大的问题，“找到问题”是关键所在，这一过程中通常包括一些能够帮助思维和具体行动的表征系统。一门语言要能允许你快速厘清和修正你对于情境和话语环境的认知，比如，能提供你所需的自动开发环境的不同工具等，并且允许你进行实际的探索。

你参与了视觉编程环境相关项目，如 GRAIL 和 Etoys，你对现在的视觉编程怎么看？

Alan：我并没有参与 GRAIL，但我一直非常钦佩那些能够做到这一点的人（当时他们尽其所能了）。很值得去回顾一下当时的历史！现在的编程（无论是否是视觉编程），对象是儿童还是成人，都很差强人意。

你认为下一代编程语言，应该具备什么特性？

Alan：十年前我就写过或讨论过相关主题。我想关于这个事情的思考，很大的一个问题在于那些应该对它进行认真思索的人都在“编程”方面有很多自己的洞见，但他们的思想过于老旧，所以这限制了他们去寻找今天对于他们而言最重要的议题。

如果我们仔细研究各种工程领域技术，比如在机械、电子、生物的 CAD、SIM、FAB 等，我们就更能发现所需元素。从另一个角度来看，如果我们发现了对设计和评估之类过程的巨大需求，我们也能发现对要求、规格、合法性的各种表征必须都是可以除障、可以运行，也能进入到一种新的 CAD、SIM、FAB 编程过程。许多即将发生的趋势都是要除掉主流的“怎么做”的困惑，取而代之用“做什么”来替代。

JavaScript 和 Python 这样的动态编程语言越来越火，它们能代表未来吗？

Alan：如果有一种类型理念能够带有很高的明晰度就再好不过了。例如，在许多维度都非常有用的“语义类型”，不是价值导向为基础的类型（我不认为后者具有多大的价值）。

对于分布式运算和面向对象编程在未来起到的作用你怎么看？

Alan: 对于我曾提到的“面向对象”与“面向系统”两个概念，最后倒向了抽象数据类型，我常常觉得诧异不已。我想这是因为人们希望使用各种程序，任务陈述以及数据架构等老旧途径进行编程。这些方法都很难集成升级，过去我们也花了很大的力气来保留各种旧的范式。

10 年前我读过 PARC 发表的一项研究，大意是将相互连接的电脑放在不同房间里，用发射器向各个房间传送信号，每个房间中的计算机监测到信号就触发该房间的任务，这算是“环境计算”吗？你还记得这个实验吗？

Alan：记得，这个实验最初源于 70 年代的 Nicholas Negroponte。PARC 沿用的版本叫做“无处不在的计算机”，随后在 80 年代由 Mark Wieser 发展。

你认为我们不需要操作系统，是否因为有了浏览器就足够了？当前，浏览器开发有赖于操作系统，你认为未来在没有操作系统的前提下，网页浏览器可以实现编译吗？

Alan：很值得思考真正的物体作为真实虚拟机器互通消息所带来的真正影响，它们的灵感部分来自于 Licklider 的视觉所构成的强有力的世界性网络。思考时可以想，“硬件目标”只是能够体现各种进程与意图的软件目标的缓存。

作为缓存，每台计算机都需要少量代码来处理时间与空间资源以及与网络之间连接的开关闭合。这可以被称为“微核”，但在客观世界，这也可以用实体来表示，而非一个“数据栈”（stack）。为了能“完成目标”，可以想象这些“真实的物体”在系统内栖息在一个或多个硬件缓存之上。具体的组合则取决于不同缓存的各个资源分配。

了不起的 Gerry Popek（《世界计算机周刊》CTO）在 80 年代进行了这类缓存结构的首次尝试，并将它应用在了不同的机器类型组合之上——当时称为 LOCUS——MIT 出版社出版了一本极好的书，解释了其中的相关细节和工作原理。当时它的主要掣肘在于它是 UNIX 进程延伸，但它的确证明了这些带有缓存资源的分散性“物体”的工作原理。（当我在 1984 年到那里的时候，我曾尝试让苹果公司买下这个系统）。

关键点（Bottom line）是没有其他东西能仿照今天大型的一体化操作系统，它能“尝试”让软件对其产生依赖，而非让软件四处被调度和挪用。因此，操作系统的存在不是太有必要。Smalltalk 就没有运行在操作系统上。

你如何看待编程中的各种范例？

Alan：一个好办法就是尝试去做你的目标中可能达到的最复杂的境界，努力实践，然后再看看是否得到了“潜在的语言”。大多数人喜欢容易上手的方法，但这就犯了大错误——这些例子实践出来都很难升级。例如，“数据”从小处看“貌似自然”，但整体框架却非常不容易升级。今天的大多数范例也是如此。

程序员：你怎么看语义网（Semantic Web），为什么它现在还没成功？

Alan：主要是意义模型的各方面都很弱，而且它本身没有多少新颖观点，只是人们尝试再造轮子而已，机器学习也没有发挥多少作用。

你怎么看 GO 语言，尤其是其简单。它可以作为其他编程语言的参考吗？

Alan：简单不是衡量编程语言的唯一标准。

依你之见，什么是衡量编程语言的标准呢？

Alan：你先说说你的看法，我来评论。

关系、时间、对象、用处。

Alan：这些都很重要，也能满足需求。我想从不同的角度出发，来看待“更有用”的事情，比如以下几点。

- 能够帮助我们思考普遍事物，包括发现问题和解决问题。关于认知论的关注，包括对“语言”发挥先天性影响的整体环境。
- 各种表征的对抗，我们能基于此尝试建立模型，创造动态的介入过程。包括数学——这也是为什么数学是多元的，真正的数学一定能满足实际所需。
- 许多领域的互联互通，包括含义和各种优化、定义与元定义、Debug、再形成等。实用性强的各种担心，往往最后会成为可以加工的人工制品。

你觉得逻辑清晰的人造语言未来会成功吗？

Alan：四五十年前我对这个话题感兴趣，现在对我而言还

是一个很有趣的领域，值得思索。

关于数据处理，你上面的“发送进程，而非消息”是什么意思？

Alan：“信息”是否有足够的“内容”来具体化到一个进程内部（因此它能够协助阐释和沟通），或者信息接收方需要完成所有工作（或者因此不得不了解关于系统巧妙升级的所有问题了）。

你怎么看 DCI（编者注：Data, context and interaction，用于实现内部存在相互通信的对象的系统，与 MVC 一样，同由 Trygve Reenskaug 发明，是 MVC 的一种补充）？

Alan：我们之前跟 Trygve 相处甚欢，也学到很多东西。很高兴看到他们的确在推进这一进程。

科学和它的未来

在计算机的发展进程中，有哪些转折点你希望能重新来过？

Alan：有，分别是以下三个。

- Intel 和摩托罗拉对 PARC 的硬件架构感兴趣，能允许极高层次的各种语言有效执行。80 年代还未出现这一场景，因此，让“从 50 年代到 60 年代的平庸想法”影响了编程，这也部分造成了：
- “我们知道如何编程”的假象盛行，这也在很大程度上阻碍了 70 年代的一些不错的软件促成更优的编程。在 80 年代，相反回退到了一些疲软无力的方法之上。现在我们都还没有从这些方法的余震中复原过来。
- 那些疲弱的举动，例如没有得到良好酝酿就产生的万维网与 Engelbart 所带来的真正有用并能满足实际所需的各种想法的对弈。

你和你同事的很多工作都成为了今天产业的基础，那么有没有一些想法是没有实现的？

Alan：我们最不现实的假设就是当接触到真正好的想法时，大多数人理应都会善用这些想法来学习并取得进步。实际上，能真正做到的只是小部分人，要面对其中不同规则就需要花费很大功夫了。以及，早期的采纳者都倾向于采纳了不太好的想法——好的想法只是为时过早了。

在计算机革命的进程中，80 年代好像停滞了一段时间，如果能回到那时候，我们该怎样做去推动呢？

Alan：“警察需要罪犯”。

“医生离不开疾病”。

在过去，那些有实力、有自信、有激情的人决定做点什么的时候，就成事了。而这一过程并非总是反映了“真正的现实所需”（通常都不是现实所需）但他们的确将事情发挥到一个境界，那些推进自己目标的人在这些事情里都看到了契机（包括“科技时尚”）。

我们所处的时代，的确有可能让此事再次发生。回顾过去，另外觉得诧异的不是真正优秀的研究者的稀缺，而是真正优秀的研究管理者的稀缺，而真正好的投资者尤其稀缺。可能听起来太简单了，但我看来，充沛的资金是一个好开端。因此，拿军功章的应该是资助者，而非研究者（试想想：好的资助者会事先出资，他们也充分了解，如果是他们运气不够好，70% 的金子几年后都变成铅条了）。

资金稀缺的年代应该是 60、70 年代吧，主要原因是那时的计算机的成本较高。

Alan：如果你用一台 1000 美元的计算机，你其实沿用了旧的计算方法。这里有一点，你可以用现在的超级计算机来做研究和开发，可以享用的资源在未来会以更廉价的方式来获取。就拿今天的薪资水平来看，今天要资助同样的研究，成本实际上更高了。

过去跟今天很相像，计算机界的大多数人在那时只是受本土的几个供应商和一些本地计算机思想流派的局限。当时要让计算机和其他工具回归真的很困难。当时不是一个疯狂的教授能对过去发挥影响，而是整个研究团体都必须发挥某种影响。

硬件遵循着摩尔定律，但软件确恰恰相反，我认为现在的软件往往难以发挥硬件的优势。

Alan：研究团体在过去是“一个不断争辩的团体”，并且知道如何“辩亦有道”。比如，不搞个人攻击，只是尝试就事论事。这样一直都进行得很顺利。你没注意到的一点不是硬件能做什么，而是如果不升级的话你能做什么。如果不能摆脱设计的桎梏，就很难穿上优化的新衣，一旦你缺失了前者，又会迷失。PARC 研究方法的关键就是在未来不需优化就能做许许多多设计。

我们怎样保证研究可以切实“命中目标”呢？

Alan：“寻找问题”是在“真实研究”中最难事情，因此还需要做很多事情来完成这一点。实际上，今天所有的投资者都希望知道他们所资助的对象会解决什么问题——因此他们削减了（甚至取消了）对探索部分的资助。

ARPA/PARC 的研究过程是“资助研究者，而非资助项目”——而时至今日，这看起来非常的不同寻常。发明电灯泡永远都比点蜡烛要好。

今天人们对科学的兴趣减少，是否是科学匮乏的征兆？

Alan：科学的匮乏是并非因为大多数人并不对科学真正感兴趣，而是他们根本不懂得科学是什么。

你是否喜欢将进程和形式作比较？

Alan：是的。中世纪任何人都可以研究物理——他们只需带一顶尖尖的帽子即可。在牛顿时代的几个世纪后，一个人必须要学会许多艰深的东西，但这非常值得，因为所取得结果能够将你带向更高的境界。

如今数据的价值被多次强调，然而数据本身是没有意义的，机器对数据的阐释被应用到了很多地方，你认为今后就不需要人类来做这类工作了吗？

Alan：之前我们笃信得所谓“古老而基本”的想法已经不再适用了，即使它们曾经一度辉煌。这里的关键点是我们可以让人去解释这个句子，并提出相应的问题，但是这两者还是分离的。对于一些重要的谈判协商，我们会直接派遣大使，而不是打个电报。

这就是客体的内涵，但看到一些真正的所需和实际需要还未被完全认识，我一直都颇为吃惊。信息及时得到阐释，也不代表该信息就不会用其他的角度来理解，但是从噪音中提取信号，就必须经过一个特定的过程。

你上面说，数据并不是一个好想法，但数据本身仅仅是

数据，你是的观点是怎么来的呢？

Alan：这里再讲一遍：数据的核心是“意义”，如果不经过加工，“数据”是毫无意义的。这里的一个角度就是去将“任何事情”都思考为“信息点”，然后去问要如何才能“接收到此信息”？

人们常常不自觉地就这样做了（而且还跟错误的版本打交道），所以他们需要更多地聚焦在具有实质意义的“信息”，而非所需的不同进程，来找到确切的信息并理解它。

Shannon 和 McLuhan 都用了虽然非常迥异但有用的范式来找到里面的关键要点。大多数人对此都懵懂无知——但对我而言，它充满了惊喜——也有几分令人沮丧——就是看到了计算机界的行内人都或多或少带点相似的天真烂漫。比如，今天大多数代码都依靠“代码以外”的编程观点，这非常令人震惊和沮丧。

你怎么看 Olive Executable Archive，他们现在的方向正确吗？

Alan：我想，在许多重要的例子中，这几乎是完成的唯一途径。这些问题由目光短浅的供应商和程序员受制于某些种类特定的计算机和操作系统所造成。与此相比，我们可以探讨一种更为简要的方式，它更有远见，这种方法在 PARC 得到了使用，不仅可以应用于未来，也可以用于处理我们在此设计和构建的不同计算机。

我曾举过一个例子，说它是 1978 年 Smalltalk 的再生（来自于施乐复印机废弃的一个磁盘组件），这个形象已经“永恒地”实现虚拟化，所以很容易就得以重新应用在实际生活里。

我又想起另外一个“尝试着升级”的例子，当构建不同系统的时候你完全可以在软件里构建一个通用计算机，比其内部的任何介质都要微细。

PC 盛行之后，数字似乎成了万金油，主宰一切。你认为未来数字会跟图像相结合吗？

Alan：纵观历史，一直并无所谓的“主流”才决定所思所想。许多人错误地认为“达尔文主义之下的过程”不断得到优化，但任何生物学家都会指出，这些过程只是“尝试更适应环境”所得。这也可以为“思考”所用，人类需要很长的时间去想象各种甚至比文化进程更宏大的思维进程。我们之前只拥有它们几百年的时间（在过去出现过一些有趣的光点），当然它们肯定不是所谓的“主流”。

好想法都需要时间去酝酿成长。因此，当主流阶层碰到了一个足够大的灾难，让其去思考如何改变而非原地转圈时，它通常都还是不会善用时间，不能做出真正的改变。在 PARC，那些自发明伊始就地位丝毫未被撼动的发明正是那些的确没有可以替代之物或者那些没有人曾经研究过的。例如：以太网、GUI、互联网相关的东西、激光打印等。

从另一方面看，编程的想法我觉得已经改善不少，但是还需要注意。第一，大多数人都认为他们已经知道如何编程；第二，Intel、摩托罗拉认为，他们也已经知道如何设计 CPU，并对研制 16 位处理器毫无兴趣，其实这原本能让 PARC 80 年代的更高层级的不同语言运行得更为顺利。

看来由于害怕在制造上冒险，硬件设计的门槛更高了。在未来这该怎么解决？

Alan：我不知道。过去的日子，对于“参数性的”并行计算解决方案（通过 FPGA 与其他可升级的硬件）的可能性并无得到发展（大多数人要么选择不做，要么只是往老旧的方向尝试）。一些 FPGA 模组（如 BEE3）也会划入刀片槽。

类似地，也没有办法去阻止新的软件用独立方法去开发（这意味着在最初建立与现有模式的连接已经慢慢变得无足轻重了，而新的研制结果也并没有这种具备严重后果的依赖性）。例如，我们可以做很多事情，尤其是朝着学习曲线的方式。或者，浏览器的 JavaScript 子集能够作为“足够快的硬件”（设计感一般）进行处理——以及只是“不经人手触碰”。（这在某个意义上很糟糕，但这的确是一个“真正不再读写的机器代码”）。

其中一个要素就是承认 box，但却不要接受 the box 是避免不了的（Part of this is to admit to the box, but not accept that the box is inescapable）。

你对 Web 并不是很看好，对于今天的分布式网络问题怎么看？

Alan：Web 发展中的一个错误在于把浏览器当做功能型应用。如果你看看现如今互联网的规模，就知道浏览器至少应该与操作系统类似，功能越少越好。只要能安全地运行压缩模型和处理资源即可。真的无法想象，20 多年过去了 CS101 上这么简单的原则依然无法实现。

以你的看法，基于现在的硬件设施基础，我们该如何开发分布式网络操作系统？

Alan：关于这一点，我建议你看看互联网本身，然后再看看 1978 年，Dave Reed 在麻省理工学院发表的博士论文，我们在 Croquet 项目里使用了许多他的观点。

现如今，经济压力给新系统开发带来了阻力，毕竟开发新软件（或硬件）比在现有的东西上升级困难得多。然而随着时间流逝，老系统越来越难满足新需求，想想在一个新系统上重新开发网页浏览器要耗费多少的人力就知道了。很多软件开发者发现它们的系统已经崩溃，但经济压力却让他们没法去优化。那么我们该去抵抗这股压力吗？

Alan：如果一开始有这种不好的感觉，那么最好不要去开发“网页浏览器”，当然也包括其他软件，因为没这个必要。我们现有的技术让网络入侵变得如此容易，后果可以以数十亿计累加，并传递到世界各处。如果我们把此现象类推到医学、疾病与卫生的领域里，我们应该做什么？我们又能走多远？

对我而言，我还是很难理解这些阻力对“真实事物”的本质与威力的影响，尤其是在用于升级我们用来改造互联网的实体事物的虚拟封装机被发明后（即我们所称的“电脑”）。

我看过你的一个演讲，你谈到你们当初是怎么发明出这些我们至今还在用的东西时候说，要把眼光放长远，至少要向前看 10 年。要看看这一段时间内哪些东西会变的真更有价值，然后把它们给造出来。现在你还这么想吗？

Alan：10 年怎么够，至少要 30 年，往前看 30 年才能摆脱现在的桎梏。但问题是怎样做到这一步呢？答案是先“到达”30 年后，在把“未来”的想法给带回来，而非直接从现在就过渡到未来。

你对这十年间计算机的发展似乎很失望，作为晚辈想问

问你怎样避免对技术产生失望情绪的？其实我对现在的发展挺满意的，至今觉得 UNIX 抽象是最稳固的。另外，对于如何避免“重复造轮子”你有什么建议？

Alan：我的确会抑郁低沉——谁不会呢——对抗抑郁的方法就是不要被它牵着鼻子走。关于你 UNIX 的观点，我们不妨尝试想想没有了“操作系统”的计算机系统会是怎样。一个基本的探索式方法就是避免“当你批评一样东西的时候，其实你就在间接地承认它的存在！”要首先看看这个东西是否值得存在！操作系统就不是太有必要。

学到真正的科学要多长时间？计算机科学难道不应该是真正的科学吗？

另外一个角度就是，以往博士不是人人都可以当的吗？最近（真的是最近！）怎么就变了呢？谁知道呢。所以产生了很多破坏（这些破坏还在发挥影响）一旦我们获得了真知，就不会让随便哪个三脚猫功夫的人去重要的事情里搅局（这是可能会发生的，因为又一波流行文化的幻觉与渴望正在出现）。

这不就等于剥夺别人学习的机会了吗？

Alan：有问题的不是学习过程，而是现有的工业革命不断令破损的代码不断扩散所带来的后果。没有其他语言比 Smalltalk 更能允许在随意的环境中从万事万物中学到东西——我想你可能已经认识到了这一点。

你相信“自愈型”软件吗？

Alan：这个目标值得一试，可行性也很高。请注意，生物学研究的组织性很强，所以可以以我们目前的进展为基础向前推进。但我们必须让互联网在许多方面都能自我修复。这需要实现动态稳定，而非只是尝试去研制出所谓完美的机器。

那么，遗传编程和机器学习对这一发展有帮助吗？

Alan：真正的人工智能的确能发挥作用（真的能进行阐释，也能进行升级配置）。但如果系统不能解释自身（大多数系统的确做不到），这的确不太好。

你认为未来五年软件开发的哪些方面将会对感知计算和人工智能产生影响？

Alan：我想这里有许多潜力（先想想十年内，你可能在头五年能获得真正有用的东西）。这当然首先要求参与和执行的意志力，市场其实并无没有要求一定要成就伟大的产品——学界或大多数出资者也没有这样要求。

“某个领域的研究环境越软，你就要越硬”。

PARC 在硬件、软件、市场、政府投资等领域取得的成果举世瞩目，回到当初，有哪些事情你觉得是即使到了今天也很难实现的？

Alan：70 年代的 PARC 是始于 60 年代的 ARPA 项目的副产物。像这种围绕特定愿景建立的均质文化在今天并没有存在，这也意味着像 HARC 这些地方 ARPA 项目应该有更多的文化元素营造。我想，HARC 的各种倡议比 PARC 更像 ARPA。

在用户界面设计中，你觉得哪些值得我们学习？

Alan：我不太喜欢今天大多数的 UI 设计。过去也不太喜欢——但如果从基本外表看过去，它们给人的感觉就更糟糕了。这里的“看过去”是指用户实际使用时的感觉。

如果单从类别来看，好像现在很多人都忙于重新发明很多 70 年代的东西。

Alan：可能连这一点也没有做到——我想 UI 已经走下坡路了。但是，也可以去看看有没有发生某种改变来实现“量变”。这已经在工程或科学等许多领域发生了。在计算机界又会怎样呢？

UI 走下坡路，能举个例子吗？

Alan：我们可以看看，如果 UI 放弃了 UNDO，不允许或没有展示多重任务，没有足够良好的引导来教育用户如何使用应用。类似的例子不计其数。

你喜欢把计算机科学称为一种科学吗？对于我来说，科学是设计到实验的过程。

Alan：我很喜欢“计算机科学”这个称谓，尤其是在进行真正的“计算机科学”研究时。科学从现象出发，建立了不同模型来解释现象，并希望探讨出更多的现象。我思考的时候，大多其实是在“设计”。在英语里，“进行科学研究”可以是一个动词。参考《火星救援》里的一句台词：我一定好好地对它科学研究一番”。（I'm going to have to science the shit out of this）

关于产品

你如何看待今天的硬件产品？硬件开发商该如何才能让硬件达到更好用的性能？尤其是不靠仿真，不靠概念。

Alan：如果你以“正确的进程”开始，最终就能一直追本溯源，就像沿着电线找到墙上的电源插头一样。如果你开始就研究插在墙上的插头，那中间的许多有用过程都会忽略掉。要慢慢寻求会解决现实问题的方法就是研制出新硬件，在这个模块化工具泛滥的今天，这很容易。

供应商可以做很多事情。打个比方：Intel 可以扩大其一级缓存，来制作出真正的 HLL 仿真器（也可以开拓其他解决方案）。现在，即插即用设备或 FPGA 能够在许多领域发挥效用。从另一个角度来说，我们也可以想出更多更好的方式去布局内存架构，尤其是应对多核芯片的当务之急。诸如此类。我们在追求“差强人意”的布局搭配这条道路上走得太远了，一些供应商让程序员充分发挥他们生产的各种 CPU，但没能从根本上解决问题。

你怎么看待从 Dynabook 脱胎而来的 iPad ？你对如今的平板电脑满意吗？

Alan：不满意。它们的计算能力和显示像素及深度都比我那时候能达到的水平要高，只是它们没有足够的“服务意识”（即使这件很简单的事，也没有做好），比如让一个不会绘图的人上手就可以绘图。今天所能得到的东西当中有太多类似的盲点了，并非所有都是必要的。

Dynabook 过去和现在都更多是“以服务为主”的想法，而非实际产品。今天的平板设备并没有很好地体现这些服务想法。

灵感、生活与其他

与许多年前做研究时相比，今天的你，在思考和研究方式上有哪些不同？

Alan：我是非常幸运的人，能够在 60 年代（根源可以追溯到 40、50 年代）进入一个发展成熟的研究社区——ARPA–IPTO，这让我受益良多。 我在 2004 年曾经为这个研究团体写

过一篇致敬文章：http://www.vpri.org/pdf/m2004001_power.pdf

如果希望向你一样，在工程学和艺术方面深耕，该怎么找到灵感？

Alan：首先，清醒的时候多做白日梦，这样才能酝酿新想法。其他的来源还包括：不要过度沉溺在你的种种奇思妙想中，那些十分善于做梦的人要么平庸要么比一般人还不如。有了想法，我通常会写在笔记本上，每次重新读到都会有新的看法。

此外，普林斯顿还流传了一个茶余饭后的段子，科学家互相比较他们的灵感产生过程。一位科学家说：“通常我都会半夜睡觉的时候想到，所以会在床边放一本笔记本”。另一位则说道：“我的灵感一般诞生于洗澡时，所以我随手放了一支油性笔，随时在墙上记下来”。爱因斯坦当时也在场，被问到自己的灵感何时迸发时，他回答：“不知道呢，迄今我只是有过两个想法！”看来，一些人的筛选就是比一般人强。

你总会梦到一些场景的。由这些场景继续往下推导，就很有趣了。

我的意思是，现在你看到的不是映射在角膜上的东西，而是呈现在你大脑中的事物（这就是为什么你睡觉时闭眼了，还能“看到”景象）。所以，我猜想，要想象到不同图像是有特定方法的，这跟那些自认为五音不全、不能辨别音调的人却能在被协助的情况下精准地辨别出音高音低是一个道理。

在你实际的工作中梦起到了多大影响？你有在梦中找到答案或灵感的经历吗？

Alan：Bob Barton 的很多很棒的想法都源于睡梦中。我的大多数灵感来自于白日梦。这个阶段大多数小孩都能轻而易举地深陷其中，但其实白日梦也能或多或少发挥重要的作用，我觉得如果一直抓住白日梦不放，好像很难真正长大，所以有利有弊咯。关于灵感，只要是灵感，它们的雏形往往都很平庸，所以我们要采取一些方法来应对这个问题。

我印象最深的就是你涉猎广泛，计算机科学外，哪些领域对你今天的成就还造成了影响？

Alan：我成长的环境里，书籍无处不在，我的父母和亲戚都是各个领域的从业者。这影响了我的求学心态，当然，其实这些条件也对我有帮助，只是我在里面挣扎取舍了。2004 年时当然我受邀写了一篇相关文章《The Center of “Why”》http://www.vpri.org/pdf/m2004002_center.pdf，你可以看下。

在你的巅峰时期，你每天睡多少小时？

Alan：过去我每天只睡 5 小时，一直持续到我 50 多岁，后来就患上了呼吸道传染病，是在坐飞机上感染的。多年以后，我的睡眠习惯得到了调整，医生告诉我每天至少睡八小时或更多。这就不容易做到了，所以我通常都会下午去补个觉。这个习惯帮我治好了传染病，但却减少了我每天的有效工作时间。

音乐对于你的工程研究有哪些影响？

Alan：实际而言，我并不是一位工程师——但你说得对，音乐装点了我的许多排比和比喻。

你是如何成为 Life Learner 的？

Alan：拥有对你有帮助的父母（或者父母有这方面的素质）。然后一往无前地执着追求求知欲（主动追求而非迫于外因）。

你怎么看乔布斯以及苹果公司的现状？

Alan：乔布斯并不是那种很善于结人缘的人，但我和他的确维持了友好的关系，直到他去世。这部分是因为我们的生活有数次紧密的交集，不仅仅因为苹果公司，也因为 Pixar，后来我尝试让他回归到真正的教育上，这也是苹果公司的目标。

不知你是否有过跟 Seymour Cray（译者注：1958 年设计建造了世界上第一台基于晶体管的超级计算机；同时也对精简指令微处理器的产生有重大贡献）共事的经历，尤其是 CDC 6600 项目。关于 Seymour 有哪些有趣的轶事可以分享？

Alan：Seymour Cray 是一个话很少的人。我跟他一起待了三个星期，才知道他不是保安。这里就没有足够去的篇幅去讲 Chippewa 操作系统了，但事实上，最终官方的数据控制团队也没能为 CDC 6600 项目开发出任何软件！从那以后，像我们这些来自利弗莫尔（美国加利福尼亚州西部城市），洛斯阿拉莫斯（美国新墨西哥州中部城镇），NCAR（国家大气研究中心）之类地方——即已经购买了这些机器的人，都汇聚在奇博瓦福尔斯（Chippewa Falls）想“做点什么”。

也许这个非官方的软件最有意思的地方在于它的多任务处理操作系统装有由 Seymour Cray 撰写的 6600 系列的图像 debug 工具，而且是八进制！我曾有幸为这个系统撰写一个反编译器，所以普通人也可以对系统进行编辑管理（这是一个奇妙的绝技之旅，因为这台主机的平行架构和多重处理进程）。这也给了 Seymour 一个绝好的发挥所长的空间，并且巧妙地避开了他的弱项（这台机器的确有一些很棒的地方，当然也有待完善之处——优点和缺点都很明显）。

在白天你是怎样集中精力的？

Alan：休息一会、弹奏乐器、冲个澡、发个呆、读读垃圾小说，等等。

你对在计算机领域保持工作生活的平衡有哪些体会？

Alan：要努力避免“变老”！行业面临的一个普遍问题就是缺少带薪或有补贴的“休假年”——而这是再充电和再创造的好方法。施乐过去给员工提供了带薪休假年，这很令人诧异。但最近几年，我都没听说过了。

如果你觉得生活需要“可再生资源”，你不应该让你自己“被榨干”，不过说起来容易做起来难。

你怎样理解“游戏化”这个概念？从提高人的注意力的角度来看，你认为密集的奖励制度和随机的奖励制度哪种更好。我想到音乐的例子，音符以某种规律排列形成“乐音”给人享受，游戏的设定是否能参照这一思路？

Alan：我显然不会按照这些原则去思考音乐，也不会这样去评定电影戏剧。 我很喜欢各种成熟的艺术，要学会这些艺术要求亲自参与感受，而不是被动地照单全收，像小狗对抛向它的每根骨头都来者不拒一样。

Bob Taylor 现在为什么不在写点类似《创新未酬》的东西了？

Alan：《创新未酬》（Dealers of Lighting）并不是最佳的推荐，你或许可以读读 Mitchell Waldrop 的《梦想机器》（The Dream Machine）。 基于这一点，怎样赞誉 Bob Taylor 都不为过，无论

是他在 PARC 的工作经验还是他早先作为 ARPA-IPTO 的负责人，他的业绩都无与伦比。

Bob Taylor 这样的人物在历史的各个阶段中都是少数。那时和现在的区别就是赞助人在那时的确做了“正确的”选择，但在最近 30 年又屡屡变得“毫无头绪”。最近一次挺有趣的另外就是 Sam Altman（YCombinator 总裁）所做的事情——这已经慢慢成为 70 年代以来最大、最有趣的尝试了。

我想跳出固有思维模式该怎么做呢？

Alan：要多努力意识到你自己，包括我们所有人都是有固有思维模式的，这意味着我们所认为的“现实”只是我们自己的认知所在，“现状”是一个具体的场景构建。未来与某个现状并非一定要有某种关联，因为它只是一个具体的构建。这就会允许你更多地用过去的经验来影响当下的思考。你可以去研究那些与现在毫无关联的事物，但它们对现在都可能发挥巨大的作用。

对于想进入你所从事的研究领域的人而言，你有什么建议？

Alan：“我比所有人都要更感激自己所在的研究团体”。50 年前，我有幸进入了 ARPA（当时其实还不了解它的存在）。好的开始就是找到那些你觉得有趣的人，有趣的领域。

变老对你来说是不是感觉很糟糕？你现在还有激情做些新东西吗？

Alan：“变老”本身不是一件糟糕的事情——你在尝试做不同事情的时候，会知道毅力有多重要。其实我们很幸运，大多数“新”事物更像新闻一样存在，而非实际上有什么新颖之处。从实际范畴变化的观点来看，过去三十年事物的发展非常缓慢。

有没有什么问题是你特想让我问，但我又没问的？而且这些问题能让你更好地表达自己的思想。

Alan：没有。实际上我认为我的真实内在就像是某种不能消化的“毛团”，所以各种问题对我都很有帮助。

Peter Norvig：人工智能将是软件工程的重要部分

文 / Peter Norvig

Peter Norvig 是誉满全球的人工智能专家，目前担任 Google 研究总监（Director of Research），他同时也是经典书籍《人工智能编程范式：Common Lisp 案例研究》（Paradigms of AI Programming: Case Studies in Common Lisp）和《人工智能：一种现代方法》（Artificial Intelligence: A Modern Approach）的作者 / 合著者。在本文中，我们将看到 Peter Norvig 对人工智能目前进展和未来发展的思考，对人工智能技术在 Google 应用的解读，以及对最新软件工程师在人工智能时代的成长的观点。

Peter Nervig 眼中的人工智能

问：人工智能领域在哪些方面发生了您未曾预料的演变？

Peter Norvig：在 1980 年我开始从事人工智能研究时人工智能意味着：一位研究生用说明性语言写下事实，然后拨弄这些事实和推理机制，直到从精心挑选的样本上得到不错的结果，然后写一篇关于它的论文。

虽然我接受并遵循这种工作模式，在我获得博士学位的过程中，我发现了这种方法的三个问题：

- 写下事实太慢了。
- 我们没有处理异常情况或模糊状态的良好方法。
- 这个过程不科学——即使在选定的样本上它能工作，但是在其他样本上工作效果会如何呢？

整个领域的演变回答了这三个问题：

- 我们依靠机器学习，而不是研究生付出的辛苦努力。
- 我们使用概率推理，而不是布尔逻辑。
- 我们希望使用科学严格的方式；我们有训练数据和测试数据的概念，而且我们也有比较不同系统处理标准问题所得到的结果。

1950 年，阿兰图灵写道：“我们只能看到未来很短的一段距离，但是我们很清楚还有什么需要完成。”自从 1950 年，我们已经得到许多发展并实现了许多目标，但图灵的话仍然成立。

问：对于机器学习研究，工业界与学术界有何不同呢？

Peter Norvig：我认为，在教育机构、商业机构还是政府机构并不是很重要——我曾经在这三种机构都学到很多东西。

我建议你在有着一群出色同事和有趣问题的环境下工作。可以是工业界、学术界、政府或者非营利企业，甚至开源社区。在这些领域里，工业界往往有更多的资源（人、计算能力和数据），但如今有很多公开可用的数据供你使用，一个小团队，一台笔记本电脑，或者一个小而廉价的 GPU 集群，或者在云计算服务上租赁或捐献时间。

问：您对深度学习有什么看法？

Peter Norvig：我清楚地记得 80 年代初的那一天，Geoff Hinton 来到伯克利进行了关于玻尔兹曼机的讲座。对我来说，这是个了不起的视角——他不赞同符号主义人工智能很强大很有用，而我了解到了一种机制，有三件令人兴奋的新（对我而言）事情：根据大脑模型得出的认知合理性；从经验而不是手工编码中学习的模型；还有表示是连续的，而不是布尔值，因此可以避免传统符号专家系统的一些脆弱问题。

事实证明，玻尔兹曼机在那个时代并没有广泛普及，相反，Hinton、LeCun、Bengio、Olshausen、Osindero、Sutskever、Courville、Ng 以及其他人设计的架构得到很好的普及。是什么造成了这种不同呢？是一次一层的训练技术吗？是 ReLU 激活函数？是需要更多的数据？还是使用 GPU 集群可以更快地训练？我不敢肯定，我希望持续的分析可以给我们带来更好的了解。但我可以说，在语音识别、计算机视觉识别物体、围棋和其他领域，这一差距是巨大的：使用深度学习可以降低错误率，这两个领域在过去几年都发生了彻底变化：基本上所有的团队都选择了深度学习，因为它管用。

许多问题依然存在。在计算机视觉里，我们好奇深度网络实际上在做什么：我们可以在一个级别上确定线条识别器，在更高层次确定眼睛和鼻子识别器，然后就是脸部识别器，最终就是整个人的识别器。但在其他领域，一直很难了解网络在做

什么。是因为我们没有正确的分析和可视化工具吗？还是因为实际上表示不一致？

在有许多数据的时候，深度学习在各种应用中表现不错，但对于一次性或零次学习，需要将一个领域的知识转移并适应到当前领域又如何呢？深度网络形成了什么样的抽象，我们可以如何解释这些抽象并结合它们？网络会被对抗性输入愚弄；我们如何预防这些，它们代表了根本缺陷还是不相干的把戏？

我们如何处理一个领域中的结构？我们有循环网络（Recurrent Networks）来处理时间，有递归网络（Recrsive Networks）来处理嵌套结构，但这些是否已经足够，现在讨论还为时过早。

我对深度学习感到兴奋，因为很多长期存在的领域也是如此。而且我有兴趣了解更多，因为还有许多剩余问题，而且这些问题的答案不仅会告诉我们更多关于深度学习的东西，还可以帮助我们大体理解学习、推理和表示。

问：在深度学习最近取得的成就之后，符号主义人工智能是否还有意义？

Peter Norvig：是的。我们围绕着符号主义人工智能开发了许多强大的原理：逻辑预测、约束满足问题、规划问题、自然语言处理，乃至概率预测。因为这些算法的出色表现，我们处理问题的能力比原来提升了几个数量级。放弃这一切是件可耻的事。我认为其中一个有意识的研究方向是回过头看每一种方法，探索非原子式符号被原子式符号取代的这个过程究竟发生了什么，诸如 Word2Vec 产生的 Word Embedding 之类的原理。

下面是一些例子。假设你有这些逻辑“事实”：

- 人会说话；
- 除人以外的动物不会说话；
- 卡通人物角色会说话；
- 鱼会游泳；
- 鱼是除人以外的动物；
- Nemo 是一个卡通人物；
- Nemo 是一条鱼；
- 那么我们要问了：
- Nemo 会说话吗？
- Nemo 会游泳吗？

用逻辑来表述和解释这个场景的时候遇到了两个大问题。首先，这些事实都有例外，但是用逻辑很难穷举这些例外情况，而且当你逻辑出错的时候预测就会出问题了。其次，在相互矛盾的情况下则逻辑无能为力，就像这里的 Nemo 既会说话又不会说话。也许我们可以用 Word Embedding 技术来解决这些问题。我们还需要 Modus Ponens Embedding（分离规则，一种数学演绎推理规则）吗？不学习“如果 A 且 A 暗示 B，则 B”这样一种抽象的规则，我们是否可以学习何时应用这种规则是恰当的？我觉得这是一个重要的研究领域。

再说一点：许多所谓的符号主意人工智能技术实际上还是优秀的计算机科学算法。举个例子，搜索算法，无论 A* 或是蚁群优化，或是其他任何东西，都是一种关键的算法，永远都会非常有用。即使是基于深度学习的 AlphaGo，也包含了搜索模块。

问：我们哪儿做错了？为什么 Common Lisp 不能治愈世界？

Peter Norvig：我认为 Common Lisp 的思想确实能治愈这个世界。如果你回到 1981 年，Lisp 被视作另类，因为它所具有的下面这些特性还不被 C 语言程序员所知：

- 垃圾回收机制；
- 丰富的容器类型及相应的操作；
- 强大的对象系统，伴随着各种继承和原生函数；
- 定义测试例子的亚语言（sublanguage）（并不属于官方版本的一部分，但我自己配置了一套）；
- 有交互式的读入 – 运算 – 打印循环；
- 敏捷的、增量式的开发模式，而不是一步到位的模式；
- 运行时对象和函数的自省；
- 能自定义领域特定语言的宏。

如今，除了宏之外的所有这些特性都在主流编程语言里非常常见。所以说它的思想取胜了，而 Common Lisp 的实现却没有 —— 也许是因为 CL 还遗留了不少 1958 年编程语言的陋习；也许只是因为一些人不喜欢用大括号。

至于说宏，我也希望它能流行起来，但当用到宏的时候，你成为了一名语言设计者，而许多开发团队喜欢保持底层语言的稳定性，尤其是那些大团队。我想最好有一套使用宏的实用指南，而不是把它们全部抛弃（或是在 C 语言里严格限制的宏）。

问：在未来 10 年里，有没有哪些情况下软件工程师不需要学习人工智能或机器学习的，还是每个人都需要学习？

Peter Norvig：机器学习将会是（或许已经是）软件工程的一个重要部分，每个人都必须知道它的运用场景。但就像数据库管理员或用户界面设计一样，并不意味着每个工程师都必须成为机器学习专家——和这个领域的专家共事也是可以的。但是你知道的机器学习知识越多，在构建解决方案方面的能力就越好。

我也认为机器学习专家和软件工程师聚在一起进行机器学习系统软件开发最佳实践将会很重要。目前我们有一套软件测试体制，你可以定义单元测试并在其中调用方法，比如 assertTrue 或者 assertEquals。我们还需要新的测试过程，包括运行试验、分析结果、对比今天和历史结果来查看偏移、决定这种偏移是随机变化还是数据不平稳等。这是一个伟大的领域，软件工程师和机器学习人员一同协作，创建新的、更好的东西。

问：我想从软件工程师转行成为人工智能研究员，应该如何训练自己呢？

Peter Norvig：我认为这不是转行，而是一种技能上的提升。人工智能的关键点在于搭建系统，这正是你手头上的工作。所以你在处理系统复杂性和选择合适的抽象关系方面都有经验，参与过完整的设计、开发和测试流程；这些对于 AI 研究员和软件工程师来说都是基本要求。有句老话这样说，当一项人工智能技术成功之后，它就不再属于人工智能，而是成为了软件工程的一部分。人工智能工作者抱怨上述观点的意思就是他们的工作永远离成功有一步之遥，但你可以认为这表明你只是需要在已知的基础上再添加一些新概念和新技术。

人工智能在 Google

问：Google“没有更好的算法，只是多了点数据而已”这种说法是真的吗？

Peter Norvig：我曾引用微软研究院 Michele Banko 和 Eric Brill 发表的一篇关于分析词性辨析算法的论文，他们发现增加训练数据得到的效果提升比更换算法更明显。我说过有些问题

确实如此，而另一些问题则不见得。你可以认为这篇论文是“大数据”的功劳，但要注意，在这个领域十亿个单词规模的训练数据集就能看出效果 —— 在笔记本电脑的处理范围内 —— 还不到数据中心的量级。所以，如果你用不了数据中心，不必担心 —— 你拥有的计算资源和数据量几乎完胜任何一个上一代的人，你可以有更多的新发现。

所以没错，大量与任务相契合的高质量数据必然会有帮助。然而真正有挑战的工作在于发明新学习系统的研究和让其真正落实到产品中的工程实现。这个工作正是大多数机器学习成功案例的驱动力。正如 Pat Winston 所说：“人工智能就像葡萄干面包里的葡萄干，葡萄干面包的主要成分还是面包，人工智能软件主体也是常规的软件工程和产品开发。”

问：成为一家“AI-first”公司对 Google 意味着什么？

Peter Norvig：“传统”的 Google 是一个信息检索公司：你提供一个查询，我们快速返回 10 个相关网页结果，然后你负责找到与查询词相关的返回结果。“现代”的 Google，CEO Sundar Pichai 设定了愿景，它不仅基于相关信息建议，还基于通知和助理。通知，意味着当你需要时，我们提供你需要的信息。例如，Google Now 告诉你该去赴约了，或者你目前在一家杂货店，之前你设定了提醒要买牛奶。助理意味着帮助你实施行动——如规划行程、预订房间。你在互联网上可以做的任何事情，Google 都应该可以帮你实现。

对于信息检索，80% 以上的召回率和准确率是非常不错的——不需要所有建议都完美，因为用户可以忽略坏的建议。对于助理，门槛就高了许多，你不会使用 20% 甚至 2% 的情形下都预订错房间的服务。所以助理必须更加精准，从而要求更智能、更了解情况。这就是我们所说的“AI–first”。

Peter Nervig 在 Google

问：你的职业生涯如何起步？

Peter Nervig：我很幸运地进入了一所既有计算机编程又有语言课程的高中（在马萨诸塞州牛顿县）。这激发了我将两者结合起来学习的兴趣。在高中阶段无法实现这个想法，但是到了大学我主修应用数学专业，得以研究这方面（当时，我们学校并没有真正的计算机专业。我开始是主修数学，但很快发现自己并不擅长数学证明，反而在编程方面如鱼得水）。

大学毕业后，我当了两年的程序员，不过仍旧一直在思考这些想法，最后还是申请了研究生回来继续从事科研（我过了四年才厌倦大学生活，而两年就厌倦了工作状态，所以我觉得我对学校的热爱是对工作的两倍）。研究生阶段为我学术生涯打下了基础，而我却迷上了今天所谓的“大数据”（当时还没有这种叫法），我意识到在工业界更容易获得所需要的资源，因此放弃了高校里的职位。我感到幸运的是每个阶段都有优秀的合作伙伴和新的挑战。

问：你在 Google 具体做什么？

Peter Norvig：在 Google 最棒的事情之一就是总有新鲜事；你不会陷入例行公事之中。在快节奏的世界中每周都是如此，当我角色改变之后，每年更是如此。我管理的人员从两人变成了两百人，这意味着我有时候能深入到所参与项目的技术细节中，有时候因为管理的团队太大，只能提一些高层次的笼统看法，并且我相信我的团队正在做的事情是正确的。在那些项目里，我扮演的角色更多的是沟通者和媒介——试图解释公司的发展方向，一个项目具体如何展开，将项目团队介绍给合适的合作伙伴、制造商和消费者，让团队制定出如何实现目标的细节。我在 Google 不写代码，但是如果我有一个想法，我可以使用内部工具写代码进行实验，看看这个想法是否值得尝试。我同样会进行代码审查，这样我就可以了解团队生产的代码，而且这也必须有人去做。

还有很多的会议、邮件、文档要处理。与其他我工作过的公司相比，Google 的官僚主义更少，但有时候是不可避免的。我也会花一些时间参加会议、去大学演讲、与客户交流，以及参与 Quora 问答。

问：在加入 Google 之前，你曾担任美国宇航局（NASA）计算科学部门的主要负责人，在美国宇航局的工作与 Google 的工作有何不同？有哪些文化的差异？

Peter Norvig：美国宇航局与 Google 有很多共同之处：它们都有一群优秀敬业并且充满激情的员工，这些人相信它们的工作使命。而且两者都在推动各自技术的上限。因此，他们在特定项目中的文化往往是相似的。

同时也存在一些差异。美国宇航局的 Gene Kranz 曾说过一句名言：“失败不是种选择（Failure is not an option）。”美国宇航局经常会有几亿美元的使命任务，任何一个错误都有可能毁灭一切。因此，需要极其小心。Google 的项目范围往往更接近 Adam Savage 的想法（与 Jeff Dean 相互呼应）“失败自古至今就是一种选择（Failure is always an option）”。Google 相信，单台计算机可能会发生故障，而设计网络系统可以从故障中恢复。在 Google，有时我们可以在用户看到错误之前进行恢复纠正，而有时当一个错误曝光后，我们可以在简短的时间内纠正它，同时向受到影响的用户致歉，而这在美国宇航局是很少见的。

一方面是因为失败的预期成本存在差异，另一方面是由于空间硬件的成本巨大（参见我在那做的东西），再者就是政府与私人机构的差异，基于这一优势，Google 更容易启动新项目，并在已有的项目中迅速推动新项目的进展。

问：你是如何权衡新功能的开发与旧功能的维护呢？

Peter Norvig：尽你所能将任务做得最好，并且不断改进，这样就会得到提高。

我曾一次次地发现：团队的新员工说“我们为什么不使用 X？”，一位老员工回答说：“我们三年前就试过了 X，结果证明它并不管用”。此时的难题是：你是否接受那位老前辈的回答？或者说，现在的情况已经改变了，是时候重新审视 X 了？也许我们有新的数据，或者新的技术，又或者新员工将采取不同的方法，或者说世界改变了，X 将会比以往工作得更好。我无法告诉你该问题的答案，你必须权衡所有证据，并与其他类似问题进行比较。

程序员提升之道

问：《人工智能：一种现代方法》还会有新的版本吗？

Peter Norvig：是的，我正在为此努力。但至少还需要一年的时间。

问：我是一名研究生，我的人工智能课程使用《人工智能：一种现代方法》作为参考教材，我如何才能为人工智能编程项目做贡献？

Peter Norvig：现在正是时候：我正在为《人工智能：一种现代方法》这本书的下一个版本的配套代码工作，在 https://github.com/aimacode 上，你可以找到 Java、Python 和 JavaScript 子项目，我们一直在寻找好的贡献者。除了提供书中所有算法的代码实现，我们还希望提供 tutorial 材料和练习。此外，GitHub 上也还有其他好的人工智能项目，都希望有铁杆贡献者。

问：有没有像可汗学院（Khan Academy）和 Udacity 一样的在线资源，可以让人们在不到“十年”就精通一门学科呢？

Peter Norvig：精通可能需要十年，或者是 10000 个小时，这种时间会因任务、个体以及训练方法的不同而有所差异。但真正的精通并非易事。可汗学院和 Udacity 主要是提供了技术培训，让你不断努力地学习直到你真正地掌握它。在传统的学校教学当中，如果你在考试中获得的成绩是“C”，你就不会再去花更多的时间去学习并掌握它，你会继而专注于下一个学科，因为班集里每个人都是这样做的。在线资源不是万能的，精通它需要加倍努力地学习，而学习需要动力，动力则可以通过人与人之间的联系逐步提升，这在网上是很难学到的。因此，在一个领域，走上正轨，我们需要在社交、动机方面做更多的工作，我们需要对个人需求有针对性地做更多的定制培训，同时我们还需要做更多使实践审慎和有效的工作。我认为，在线资源主要的最终结果不是缩短精通的时长，而是增加更多学生实现精通的机会。

问：如果请你再次教授《计算机程序设计》（Udacity）这门课程，会做哪些改变呢？

Peter Norvig：我认为这门课程很好，反馈（不管是数量还是质量）大多都是好的。就个人而言，我希望有更多的实例程序和技术。我想修正之前我们犯下的一些错误（主要是因为课程进展太快，没有太多的时间去测试所有的东西）。我希望系统能够更加互动：让学生获得更多的反馈信息，不仅仅是“你的程序不正确”，同时可以让学生看到下一件要做的事情，让他们知道到目前为止已经做了什么。我认为对于学生而言，正则表达式和语言这部分进展速度过快了；另外，我还想添加更多的材料，让学生加快学习速度，同时给他们更多的机会去实践新想法。

MINIX 30 年之经验教训谈

Linux 已经为众人所熟知，而它的直接始源——MINIX 到现在也有 30 年历史了，如此高龄的软件却还依旧充满生机。MINIX 的故事，以及它和 Linux 的出现历史流传并不广泛，也许从 MINIX 的开发上我们能学到一些经验教训。其中有些是关于操作系统的，有些是关于软件工程的，还有一些是关于其他领域的（比如项目管理）。无论 MINIX 还是 Linux 都不是凭空而来的。两者的出现都有很多的相关历史，因此为了正确理解本文，我们先要对它做个简单的介绍。

要点

- 每个设备驱动都应当按照独立的用户模式进程来运行。
- 软件可以维护很长时间，在设计时应当有前瞻性。
- 想让人们接受全新、颠覆性的概念十分困难。

1960 年，在麻省理工学院有一个跟房间差不多大小的真空管科学计算机，名叫 IBM709。尽管远远比不上如今的电脑——苹果 iPad 比它快 7 万倍，RAM 也比它大 7300 倍——但 IBM709 已经是那时世界上最强大的电脑了。使用者一般在 80 列穿孔卡片上用 FORTRAN 来编写程序，并将这些卡片拿给操作员执行人工读取。几个小时之后，打印在 132 列折叠式记录纸上的结果出炉。在 FORTRAN 语句中就算漏掉一个逗号，都会导致编译错误，而致使程序员几个小时的时间付诸流水。

为了给用户提供更好的服务，麻省理工开发了一套新系统，叫做兼容分时系统（CTSS）。用户可以使用交互式终端来操作，等待时间也从数小时减到数秒，同时后台还能运行旧式的批处理任务。在 1964 年，麻省理工、贝尔实验室联合那时还是电脑供应商的通用电气公司，一起参与开发了它的后续系统，这个新系统能够同时为波士顿地区的数百名用户提供服务。我们可以把它当作云计算 0.0 版本，这个系统被称为 MULTICS（MULTiplexed Information and Computing Service）。长话短说，MULTICS 在初期问题很多，第一个版本所需求的 RAM 就超出了 GE 645 的 288KB 内存。最后在改进了 PL/1 编译器之后，MULTICS 终于得以启动运行。然而，贝尔实验室很快就失去兴趣并退出了这个项目，留了一个开发人员 Ken Thompson。Thompson 有着勃勃雄心，他希望能在廉价的硬件上复制一个缩减版的 MULTICS。在 1973 年 MULTICS 投入商用后，它在全世界揽收了大量的安装份额，最后停止使用的记录是在 2000 年 10 月 30 日，之间整整运行了 27 年。

回到贝尔实验室之后，Thompson 找到了一台迷你电脑 PDP-7，他在这台电脑上用汇编语言编写了 MULTICS 的删减版。由于每次只能服务一名用户，Thompson 的同事 Brian Kernighan 将其称为 UNICS（UNIplexed Information and Computing Service）。尽管这个称呼带有讽刺意味的双关含义，象征着 MULTICS 被阉割成了 UNICS，却还是沿用了下来，不过之后逐渐被拼为 UNIX。由于已经不能算是缩略词了，有时候也写作 Unix。

1972 年，Thompson 与贝尔实验室的另一名同事 Dennis Ritchie 组成了搭档，并为 Unix 作了一个编译器，而 Ritchie 正是 C 语言的设计者。他们一同在 PDP-11 迷你电脑上用 C 语言实现了 UNIX。后来，UNIX 又出了几个内部版本，直到 1975 年贝尔实验室以 300 美元的价格将 UNIX V6 权授给大学使用。由于 PDP-11 大受欢迎，UNIX 也很快在全世界流行起来。

在 1977 年，悉尼新南威尔士大学的 John Lions 在 V6 的源代码中撰写了注释，逐行解释其含义。全世界有数百所大学开始使用 Lions 的书作为 UNIX V6 教程。

当贝尔实验室的拥有者听说有数千名学生在学习他们的产品时，全都大吃一惊——This had to stop. 在下一个版本中（1979

年的 V7），官方明确禁止任何人用 Unix 写书或者给学生授课，导致操作系统课程回归到单纯的理论授课或模拟器模式，全世界的教授为之沮丧不已。UNIX 的早期历史在 Peter Salus 1994 年的书中都有记录。

MINIX的创建

直到 1984 年我决定利用业余时间改写 UNIX V7，彼时我在阿姆斯特丹自由大学授课，希望这次改写能给学生们提供一个 UNIX 兼容的操作系统。我为这个名叫 MIni-uNIX 或者 MINIX 的系统设定的目标是 IBM 的新款 PC，因为这款机器足够廉价（1565 美元起），学生们买得起。由于早期的 PC 不带硬盘，在我的设计中兼容 V7 的 MINIX 要求配置是：256KB RAM 的 IBM PC，加一个 360KB 5 寸软盘。这个配置要远远低于 PDP-11 V7 的要求。尽管建议配置很低，但我从一开始就知道，为了编译构建整个系统，我需要更好的配置：尽可能达到最大的 RAM（640KB）以及 2 个 360KB 的 5 寸软盘。

MINIX 的设计目标如下：

- 构建运行在IBM PC上的V7克隆，载体是一个360KB的软盘；
- 将系统设计为“自托管”，即自动执行构建和维护；
- 所有源码完全公开，人人都能拿到；
- 保持干净的设计，方便学生理解；
- 让（微）内核尽可能保持较小，因为内核故障是致命的；
- 将操作系统的其他部分分成独立的用户模式进程；
- 保持Hide interrupt处于很低的水平；
- 只用传递明确协议的同步消息来通信；
- 尝试让系统端口容易移植到未来的硬件中。

最初开发的时候，我用自己的家用 IBM PC 运行着与 Mark Williams 一致的 V7 克隆版，这份系统由加拿大滑铁卢大学的一名毕业生编写，但源码是不公开的。最初使用一致的版本很有必要，因为我没有 C 编译器。当程序员 Ceriel Jacobs 根据我自行研制的阿姆斯特丹编译器套件移植了一个 C 编译器后，系统就可以自托管了。因为使用 MINIX 编译，我对任何 Bug 和缺陷都非常敏感。建议所有开发者都尽可能早地亲自尝试自己开发的系统，这样就能发觉用户遇到的问题。

经验：自己试用自己的产品。

微内核确实很小，里面只有调度器、低级别的程序管理、进程间通信与设备驱动。尽管设备驱动也编译到了微内核的执行程序中，但实际上它们是独立运行的。微内核也被编译为独立运行的可执行程序。而运行系统的每个其他组件，包括文件系统与内存管理器在内都是作为单独的程序，在单独的进程中运行。由于我使用的机器（IBM PC：CPU 8088）缺少内存管理单元（MMU），虽说可以采取捷径将所有内容打包到一个可执行文件中，但为了让这个设计在有 MMU 的 CPU 上也能运行，我决定不用这个方案。

大约花了两年时间，只用晚上和周末，我终于让这个系统运行起来了。但是运行一个小时就会毫无缘由的崩溃，也没有可辨识的模式。裸机 Debug 调整操作系统，问题仍旧防不胜防，我几乎想要放弃这个项目了。

然后我做了最后一次努力。我写了一个 8088 模拟器，在上面运行 MINIX，这样崩溃时我就能获得正确的 dump 文件和堆栈追踪。结果吓了一跳，使用模拟器运行 MINIX 毫无问题，数日甚至数周还在正常运作，一次也没有崩溃。然后我完全的迷惑了。我对一名学生 Robbert van Renesse 提到了这个奇怪的情况，他告诉我传说 8088 会在过热时产生中断。虽然在 8088 的文档中没有这样的记录，他还是坚持有这样的说法。因此我插入了代码来捕捉中断。一个小时之内，我在屏幕上看到了这条信息：“你好，我出现中断了，这条信息不会再次出现。”我立即制作需要的补丁来捕捉中断。之后 MINIX 就能正常运行并准备发布了。

经验：别盲目相信文档，文档有可能是错的。

Van Renesse 随口一句对三十年后的今天影响巨大，如果他没提过中断的事情，也许我已经在绝望中放弃了。如果没有 MINIX，很难想象 Linux 的出现——Linus Torvalds 在详尽学习了 MINIX 的源代码之后，用它为基础写出了 Linux。没有 Linux，也就没有基于其上构建的 Android 系统。没有 Android，苹果和三星的股价可能会与今天有很大的不同。

经验：听取学生的意见，他们可能比你知道得更多。

大部分的基础工具都是我自己写的，MINIX 1.1 包括有 60 的工具，从 ar 到 wc，通常大小约在 4kB 左右，现在的引导装载程序要比它们大 100 倍。MINIX 的所有内容，包括二进制文件和源代码刚好能放在 8 张 360KB 的软盘中。其中四张是启动盘、根文件系统、 /usr 和 /user （见图 1）。另外四张包含全部的操作系统源代码，还有 60 个工具源代码。只有编译器的源码没有放进去，因为实在太大了。

图 1　这是四张原始 5 英寸的 MINIX 软盘

经验：微软的前 CTO Nathan Myhrvold 的观点是正确的：软件就是煤气，会扩张至充满整个容器。

开发人员想要打破这种“规则”也是有可能的，但必须付出很大的努力。默认规则就是“更加膨胀”。

找到分发源码的办法是个大问题。1987 年几乎没人联网，我决定写一本讲这个源码的书，就像 Lions 之前做的那样。这本书由 Prentice Hall 出版发行，所有源代码都作为书籍的赠品。在讨论之后，Prentice Hall 同意以打包的形式来出售这本书，包括 8 张 5 英寸软盘，以及 500 页的手册，售价为 69 美元，基本上与制作成本相当。Prentice Hall 并不理解什么是软件，他将软

件按成本出售的行为当作增加书籍销量的手段。后来在高可用的 1.44MB3 英寸盘出现时，我又做了一个新的 3 英寸版。

图 2　四个不同平台的 MINIX 1.5

经验：无论你的产品多有需求，首先需要有推广或分发的方式。

在发布数日后，USENET 新闻组 comp.os.minix 就开启了。不到一个月就获得了 4 万名读者，考虑到能访问 USENET 的人并不太多，这个数字非常可观。MINIX 立即受到了推崇。

很快我就从现代计算机文化书店的联合创始人 Dan Doernberg 那里收到了一封邮件，邀请我去硅谷做个 MINIX 相关的演讲。刚好我要去旧金山湾区参加一个会议，就接受了邀请。我以为他在店内放套桌椅让我签售呢，没想到他租了圣塔克拉拉会议中心，还做了广泛的宣传，整个会议中心满都是人，演讲过后的答问环节持续到午夜。

之后我收到数百封邮件，要求添加这个或那个功能。由于担心系统过大，需要更昂贵的硬件才能运行，超出学生们的预算，有一些请求被我拒绝了。很多人包括我在内，都希望 GNU/Hurd 或者 BSD 承担开源系统的重任，以便我能继续专注于教学。

人们也提供了很多建议，其中有些非常有用。在这许多贡献者中，有一个名叫 Jan-Mark Wams 的人，他编写了一个非常有用的测试套件，帮助系统 Debug。他还编写了一个新的压缩程序，比那时的很多现有程序要优秀，然后系统的发布内容减到了两张盘。即便后来有了在线发布版，软盘版也非常重要，因为有 56kbps 猫的人很少。

经验：大小有别。

1985 年，Intel 发布了 386 处理器，配合全保护模式的 32 位架构。在很多用户的帮助下（特别是澳洲的 Bruce Evans），我发布了 MINIX 的 32 位受保护模式版本。尽管当初 8088 只有一个模式，但由于我对未来的硬件早有预见，从第一天起新版本的代码就被清楚的分为能在“核心模式”中运行的，以及能在“用户模式”中作为独立进程运行的。等到 386 出现时，我们因此受益良多。另外，在原始代码中我明确地将物理地址与虚拟地址区分开来，虽然在 8088 中没有什么作用，但在 386 上起了很大作用，使得移植更为简单。此时在阿姆斯特丹自由大学还有两个人：Kees Bot 和 Philip Homburg 制作了 32 位有虚拟内存的版本，不过我决定坚持使用 Evan 贡献的那个版本，因为它更接近原始设计。

经验：尝试在设计时符合未来硬件的需求。

到 1991 年，MINIX 1.5 被移植到了 Apple Macintosh、Amiga、Atari 和 Sun SPARCstation 上，以及其他一些平台。

经验：如果不依赖硬件的特殊性能，移植到新平台时会更简单。

随着系统的发展，总有意想不到的地方出现问题。其中跟网卡驱动相关的特别讨厌，因为无法调试。最后有人发现，网卡与所标注的规格不符。

经验：跟软件一样，硬件也会有 Bug。

硬件“特性”有时可以被视为硬件的 Bug。在意大利主流电脑供应商 Olivetti 将 MINIX 移植到 PC 上时，由于无法解释的原因出现了问题，最后发现是因为 Olivetti 键盘上的几个键与 IBM 键盘所返回的扫描代码不同。这让我想到，很多国家有自己的标准化键盘，因此我对 MINIX 做了修改，让它对多个键盘提供支持，并在安装系统时可作出选择。对于意大利、法国、德国等国家的人来说，这种修改非常有用。

经验：当生命给你又酸又苦的柠檬时，你可以把它做成又甜又好喝的柠檬汁。遇到不如意，你可以让它变成好事。

Linus Torvalds买了一台PC

1991 年 1 月 5 日，默默无闻的芬兰赫尔辛基大学学生 Linus Torvalds 做了一个重要的决定。他买了一个速度飞快（33MHz）、容量很大（4MB RAM，40MB 硬盘）的 PC，主要是为了运行和学习 MINIX。在 1991 年 3 月 29 日，Torvalds 在 USENET 新闻组（comp.os.minix）发布了第一条消息：

“大家好，我已经运行 minix 一周了，现在升级到 386-minix（很好用），准备下载 gcc……”

他的第二个帖子是在 1991 年 4 月 1 日发布的，回了别人一个简单的问题：

“RTFSC（阅读源代码），里面满是注释和解决方案，应该非常清楚……”

这个帖子证明了在 10 天内，Torvalds 就彻底地学习了 MINIX 的源代码，并对没学过源代码的人表示了鄙视。MINIX 那时的目标就是要让学生很容易的学习。在 Torvalds 的个例中，结果显然很成功。

然后到了 1991 年 8 月 25 日，Torvalds 又发了一个帖子：

“致使用 minix 的人，你们好，我正在为 386（486）AT 制作一个免费的操作系统（只是个人兴趣，不会像 GNU 那样大，也没那么专业）。这个系统是从 4 月开始准备的，现在已经准备完毕。我希望喜欢 / 不喜欢 MINIX 的人都能给我些反馈，因为我的操作系统有些类似，除了其他的，在文件系统中也有着同样的物理布局。”

第二年，Torvalds 在继续学习 MINIX，并使用它开发了自己的新系统。这就是第一代 Linux 的内核。在 Linux 的核心中，仍能看到 MINIX 的文件系统和源树布局的残余痕迹。

在 1992 年 1 月 29 日，我向 comp.os.minix 发布了一条消息，

表示微内核除了性能之外都比单体设计更优秀。这个帖子引发了网路论战，即便到了 24 年后的今天，全世界也有很多学生给我发消息说明自己的站队方向。

经验：传说大象记忆力超群，而互联网就像一头大象，拥有过目不忘的本领。

当心你放在互联网上的东西，有可能在数十年后仍会出现在你面前。

对一些人来说，性能更重要。Windows NT 就设计成了微内核的模式，但后来由于性能不够，微软转向了混合型设计。在 Windows NT、2000、XP、7、8 和 10 之中，底层都有一个硬件抽象层来隐藏主板的差异，在那之上是处理中断的微内核、线程调度、低级进程间通信以及线程同步，再之上是 Windows Executive，包括一群负责流程管理、内存管理、I/O 管理、安全性等功能的独立组件，共同组成了操作系统的核心。就像 MINIX 那样，它们通过定义明确的协议来通信，不过在 MINIX 中使用的是用户进程。NT 及后继系统都属于混合型系统，所有这些组件都在内核模式中运行，环境切换较少，性能更好。因此，从软件工程师的角度来看，这属于微内核设计，但从可靠性的角度来看还是单体架构，任何组件中的某个 Bug 都会导致整个系统崩溃。苹果的 OS X 也是类似的混合型设计，底层是 Mach 3.0 的微内核，其上层（Darwin 系统）脱胎于 FreeBSD。

同样值得一提的是嵌入式计算的情况。在嵌入式计算世界中，可靠性要比性能更重要，因此微内核成为主流。QNX 是一个类 UNIX 的商用实时操作系统，广泛用于汽车、工厂自动化、发电厂与医疗器械行业。L4 microkernel11 运行在无线射频芯片上和安全处理器之上，前者已有全世界 10 亿多台手机在使用，后者也有 iPhone 6 等的新型 iOS 设备在用。L4 很小，其中一个版本只有大约 9000 行 C 代码，对于单体系统来说简直不可思议。不过由于历史问题，在微内核上仍有争议，而且性能也有些低。

关于 MINIX 的新研究是尝试在微内核之上建立容错的、多服务器的、兼容 POSIX 的操作系统。

1992 年在新闻组 comp.os.minix 里我也指出了这一点：将 Linux 与 386 架构紧密结合不是个好主意，因为 RISC 机器很快就会成为市场主流。在很大程度上，这也是正在发生的事情，有超过 500 亿（RISC）ARM 芯片上市。大多数智能手机和平板电脑都使用了 ARM CPU，包括高通骁龙、苹果 A8 和三星 Exynos 所使用的变种。此外，64 位的 ARM 服务器和笔记本也开始出现。最终 Linux 被移植到 ARM 上，不过要是一开始没有跟 x86 架构绑得太紧，移植应该更容易一些。

经验：不要以为如今的硬件永远能占据主流市场。

也是在这个关键的时候，Linux 与 GCC 编译器绑定在一起。

经验：标准出现时（比如 ANSI），请紧跟标准。

除了 Linux 的真正出现，在 1992 年还有一个很大的发展。AT&T 起诉 BSDI 公司与加州大学，声称 BSD 包含了 AT&T 的代码片段，此外 BSDI 的电话编号 1-800-ITS-UNIX 违反了 AT&T 的知识产权。这个案子到了 1994 年达成庭外和解，BSD 被判有罪，中间给了新的 Linux 系统很大的发展空间。如果 AT&T 作出更明智的选择，购买 BSDI 作为市场营销部门，则有着这样强大而成熟的对手，Linux 可能永远不会有机会崛起。

经验：如果你是世界上大公司的老板，有小的创业公司出现在你关心却一无所知的领域，就把它买下来。

在 1997 年，已经兼容 POSIX 的 MINIX2 发布了，我撰写的《操作系统设计与实践》一书也更新到第二版，合著者是 Albert Woodhull。

到了 2000 年，我最终说服了 Prentice Hall 发布 MINIX 2，并在网上免费发布所有源代码。这件事本该早些做，特别是原始协议允许大学进行无限的复制，而且卖给出版商的时候也是成本价格。

经验：在采取策略后，应当时不时回头检视一下。

作为研究项目的MINIX

MINIX 2 又缓慢发展了几年，但到了 2004 年方向出现了巨大转变，那时我从荷兰科学研究组织收到了一笔拨款，可以将纯粹的爱好变为严肃的、有资金资助的研究项目。在 2004 年之前，我都没有外部资金来源。后来我又陆续收到几笔赠款，总共 300 万美元，作为研究如何将 MINIX 打造为可靠操作系统的经费。

经验：对重要的东西做研究可以让你获得研究经费，即便它游离于主流之外。

当然，MINIX 并非唯一关注微内核的研究项目。我们后来又做了些研究，包括运行时替换崩溃的组件，这是自修复系统的第一步。事实上，现在 MINIX 已经可以做些事情了：无需重启，替换一些崩溃的核心操作系统组件，不会影响应用进程。没有其他团队能做到这一点，这为我们的团队树立了信心。

经验：早期成功能够鼓舞士气。

这种变化能实现最小权限原则，在像 Windows 或 Linux 这样的单体系统中，如果出现故障则驱动有权删除磁盘内容；在 MINIX 中，微内核的设计不允许这种做法。

经验：每个设备驱动都应当以非特权模式，按照独立用户模式方式来运行。

微软显然很了解这一点，为 Windows XP 以及后续系统引入了用户模式驱动框架。并鼓励设备驱动按照用户模式的方式来编写，就像 MINIX 那样。

2005 年，我被邀请参加 ACM 的操作系统准则研讨会，这是操作系统研究的最高荣誉了。我打算在演讲上正式宣布 MINIX 3 发布，中途我脱掉了上衣，露出有 MINIX 3 字样的 T 恤。从这天起，MINIX 网站正式允许下载。我本以为会议期间，服务器能够承受负载。由于我是会议的嘉宾，所以住在皇家套房中，房间很棒，有很漂亮的海景，但很不幸，这是唯一一间没有互联网接口的房子。更糟的是酒店没有 Wi-Fi，幸好有好心的会议组织者愿意跟我换房，我才能够在普通房间内连网处理工作。

经验：专注于你真正的目标。

在有些东西看似很美好的时候，不要分心；它说不定是一个障碍。

到 2005 年，MINIX 3 已经成为了很严肃的系统，但有很多人读过我的书，还有很多在学校里学过它，很难说服他们这个系统跟之前的“玩具”系统有巨大的不同。结果很讽刺，我有一个知名的系统，因为它之前的名声却得努力让大家认真对待它。微软很聪明，早期 Windows 包括 95 和 98 都只是 MS-

DOS，但后来他们将新系统改名为 Windows。

经验：如果你的产品第三版在主要方式上与第二版不同，干脆换个新名字。

2008 年，MINIX 获得了另一次机会，拿到了大约 350 万美元的资助。

但这个巨大的机遇也创造了一个严重的问题，我雇了 4 个专业的开发者来开发 MINIX 3，并资助了 6 名博士生和一些博士后推动研究。结果博士生与开发者之间出现了巨大分歧，博士生复制了 MINIX 3 的源码，并开始用自己的研究进行大量的修改，同时开发者忙着改进 Bug，并让代码产品化。两到三年后，我们没办法将损坏的版本恢复了。

尽管我也希望将研究结果运用到产品中，但遭到了程序员的强烈抵制。因为他们已经详细优化过代码，不希望再添加很多未经测试的东西进去。只能再做很多努力，才能让小组的研究成果转为现实产品。

经验：同时做研究和开发软件产品的话，结果会很难结合。

有时研究人员与开发人员会遇到同样的问题，其中一个涉及同步通信的使用。同步通信从一开始就存在，也非常简单，但它与可靠性的目标相矛盾。如果客户端进程 C 发送消息到服务器进程 S，然后 C 在收到回应前陷入崩溃或者无限循环，S 因为无法发送回复就会挂起。这个问题是同步通信所固有的。为了避免，我们不得不引入虚拟端点、异步通信等，导致简洁度大幅下降。

经验：爱因斯坦说的没错，事情应当尽可能保持简单，但不能太过简单。

他的意思是：每个人都应当追求简洁，并确保解决方案足够全面地进行工作，但不可要求太甚。这也是 MINIX 从最开始的指导准则。

2011 年左右，为了提高产品的专注度，我们做出了两个重要的决定：第一，我们认识到要想让人们使用系统，就得有应用，因此我们从 BSD 中借鉴了很多内容。实际上，我们已经在容错性更高的子架构中重新部署了 NetBSD 的用户环境，带来的收益是：突然多了 6000 个可用的 NetBSD 包。

经验：如果你希望人们使用你的产品，那它必须有用。

第二，我们意识到想要赢过 Windows、Linux、OS X 等桌面系统，得打一场硬仗。所以我们将 MINIX 3 移植到 ARM 处理器上，并开始专注于高可用性的嵌入式系统。此外，很多工程师都在寻找能够嵌入新相机、电视机、数字视频录像机、路由器等其他产品的操作系统，尤其是 MINIX 3 可以在使用 ARM Cortex-A8 处理器的 BeagleBone 系列单片机上运行（参见图 3）它们基本上都是完整的 PC，零售价是 50 美元，而且经常被用作嵌入式系统原型。所有都是开源硬件，弄清工作原理很容易。

图 3　BeagleBone 组件

经验：如果在市场推广中，计划 A 不起作用，那么改成计划 B。

回顾一下：微内核也许是实现高稳定性和自修复系统的最好方式，但令人惊讶的是，30 年来 MINIX 微内核几乎没有增加代码。甚至有一些主要的软件组件，包括驱动和大部分调度器都被移出。想要进行颠覆性的改变，需要耗费大量的时间，举个例子，根据微软的统计，到 2016 年 3 月仍有 2.5 电脑运行着过时的 Windows XP。

经验：我们很难改变已经根深蒂固的做事方式。

此外，现在的电脑性能如此强大，效率已经不那么重要了。例如 Android 是用 Java 编写的，比 C 要慢得多，但好像没人在意这一点。

另一个在 MINIX 中运行良好的机制就是事件驱动模型。每个服务器和驱动都包含：

```
{ get_request();
  process_request();
  send_reply();
}
```

这种设计使得测试和隔离 Debug 都很容易。

另一方面，MINIX 1 的简单性限制了它的可用性，缺乏内核多线程和请求式页面调度等在 256kB IBM PC 上并不实用的，后来这些功能可以加进去了，但我们之前没那么做，导致现在还在买单——移植某些软件非常困难。

尽管资助暂且告一段落，但 MINIX 项目还未结束。它就像很多其他项目一样，逐渐转向开源。